个性化你的阅读

编程狂人

Programming Madman

NO.9

推酷

关于推酷

推酷是专注于 IT 圈的个性化阅读社区。我们利用智能算法，从海量文章资讯中挖掘出高质量的内容，并通过分析用户的阅读偏好，准实时推荐给你最感兴趣的内容。我们推荐的内容包含科技、创业、设计、技术、营销等多方面内容，满足你日常的专业阅读需要。我们针对 IT 人还做了个活动频道，它聚合了 IT 圈最新最全的线上线下活动，使 IT 人能更方便地找到感兴趣的活动信息。

关于周刊

推酷周刊是专为 IT 人打造的行业技术周刊，目前推出的《编程狂人》是献给 广大的程序员们。 我们利用技术挖掘出那些高质量的文章，并通过人工加以筛 选整理出来。 每期的周刊一般会周一的某个时间点发布。

联系我们

tuicool2012　　164644910　　推酷网

下载 APP

Android版本

立即下载 Android

iPhone版本

免费下载就在 App Store

2014/01/20/第九期

目录

后端架构

程序人生

正文

[业界新闻]

Java 8 最终发布日期敲定，即使有 bug 也要发布

Oracle 公司计划于 2014 年 3 月 18 日发布 Java 8，这一日期已经敲定，即使届时该版本中仍存在一些小的 bug（“非致命”bug），也要按计划发布。

这一消息来自 Oracle 公司的 Java 8 版本发布经理 Mathias Axelsson，他在周一的邮件列表中表示，在发布日期到来之前，将优先考虑修复“致命性 bug”，“非致命性”bug 将考虑在后面的升级版本中进行修复，不会影响到新版本的发布日期。

目前 Java 8 开发团队正在紧锣密鼓地修复该版本中的已知 bug，以便确保能够按照原计划在 1 月 23 日发布一个 RC（候选）版本。

只要是软件，都会存在 bug 的，比如，在 2011 年 Java 7 刚发布的时候，就被曝出编译器存在 bug。此外，客户端 Java 安全问题不断。Oracle 公司最初计划在 2013 年 9 月份发布 Java 8，之所以被推迟到 2014 年，安全问题是其中一个重要的原因。

好在 Oracle 公司已经承认这些 bug 并在努力解决问题，今天 Oracle 公司发布了 147 个安全补丁，其中有 36 个是针对 JavaSE 7 的，这些补丁包含在 Oracle 公司最新发布的 JDK 7u51 版本中。

[b]导致 Java 8 推迟发布的另一个原因是 Lambda 项目的开发进度。Lambda

是 Java 8 中最重要的改进之一,其目的是使 Java 更易于为多核处理器编写代码。它为 Java 语言增加了 lambda 表达式、默认的方法以及方法引用，并扩展了库，以支持流数据的并行化操作。目前该项目已经开发完毕。

此外，Jigsaw（标准模块系统）原本也要包含在 Java 8 中，但由于开发工作相对滞后，被推迟到 Java 9 中。

原文链接：http://www.iteye.com/news/28709?utm_source=tuicool

Rust 0.9 发布，改进了线程模型

随着 Rust 语言向1.0里程牌的迈进,这一 Mozilla 支持的系统编程语言发布了0.9版，带来了许多改进。在演变为一门准备长期支持的稳定语言的过程中，Rust 已经发生了显著的变化。Rust 创建人 Graydon Hoare 说，该语言的目标用户是“沮丧的 C++开发人员”,因为它专注于成为 C/C++的现代化替代品这一目标。

Rust 是一门开源语言，它提供了一个用于 Windows 的二进制安装包，以及一个源代码包，用于基于 Unix 的系统（FreeBSD、Mac OS X 和 Linux）。

0.9版本包含了几个特性：

- Rust 现在为开发人员提供了选择，他们可以选择是构建动态链接库，还是静态链接库。
- 本地库现在成了一等公民，Rust 库的构建和分发可以不需要本地库的参与。
- I/O 基础设施经过了彻底修改。从逻辑上讲，所有的 I/O 功能现在都位于 **std::io** 模块中。通信模块（提供高级的通信抽象）也已经重写。
- 若干 I/O 变化是源于两个新库 libgreen 和 libnative 的创建。Rust 标准库不再设置成一个特定的调度方法，所以程序可以以 m:n（m 个应用程序线程映射到 n 个内核线程）或者1:1（一个应用程序线程映射到一个内核线程）模式运

行。这就允许开发人员为其应用程序选择能够提供最好性能的线程模型。

- Rust 开发人员应该注意，不要使用“托管指针（managed pointers）”（由@符号表示）以及使用了 Rc（引用计数指针）或者 Gc（垃圾收集指针）的“转换代码（transition code）”。

读者可以查看 Rust 0.9的官方发布说明来了解完整的细节信息。除了官方提供的 Rust 教程外，想要进一步学习这门语言的开发人员还有几个不同的资源：弗吉尼亚大学本科操作系统课程教授 Rust；Steve Klabnik 最近准备了“30分钟的 Rust 介绍”。

原文链接：http://www.infoq.com/cn/news/2014/01/rust09?utm_source=tuicool

oos 0.2.1 发布， C++ 的 ORM 框架

oos 0.2.1 发布，这是个 bug 修复版本，修复了 Ubuntu 13.10 的构建问题；移除了不必要的代码；改进了错误处理。

OOS 是一个 C++ 的 ORM 框架。旨在封装所有数据库后端功能，并对开发者提供统一的访问 API。使用 OOS 我们不需要了解后端数据库的类型和 SQL 语句，提供类 STL 的 API 和所有持久化对象的容器。

特性：

封装所有数据库后端

封装了 SQL 语句和数据库结构

为所有对象提供一个容器

类 STL 接口

简洁直接的设计

支持事务处理

内部引用计数机制

可用于过滤的简单表达式

支持的数据库：**SQLite**，**MySQL**

支持操作系统：**Windows**，**Linux**

无需依赖其他第三方库

易用

示例代码：

```
#include "object/object_ptr.hpp"

#include "database/session.hpp"
#include "database/transaction.hpp"

#include <exception>

oos::session db(ostore, "sqlite://person.db");

db.create();

typedef oos::object_ptr<person> person_ptr;

// insert object
person_ptr p = db.insert(new person("Theo"));

oos::transaction tr(db);

// start transaction
try {
    tr.begin();

    ostore.insert(new person("George"));
    ostore.insert(new person("Jane"));
    ostore.insert(new person("Tim"));
    ostore.insert(new person("Walter"));

    tr.commit();
} catch (std::exception&) {
    // an error occurred: do rollback
    tr.rollback();
}
```

原文链接：http://www.oschina.net/news/47886/oos-0-2-1?utm_source=tuicool

中国自主研发 cos 操作系统

1月15日，在北京我们中国终于发布了一款我们自主研发的操作系统，他有着一个响亮的名字 COS，是中国操作系统(China Opreating System)的英文首字母缩写。

我国自主知识产权智能操作系统 COS 发布

又一款中国自主开发的操作系统1月15日在北京发布，这款操作系统有着一个响亮的名字 COS，是中国操作系统(China Opreating System)的英文首字母缩写。COS 是由中国科学院软件所和上海联彤网络通讯技术有限公司共同发布的，后者是一个于2012年成立的技术公司，据悉该公司投资方背景深厚。上海联合投资有限公司是 COS 的联合发布方和商业运作的主导方。

原文链接：http://news.yesky.com/hot/332/35804832.shtml?utm_source=tuicool

[前端开发]

Javascript 和 CSS 浏览器兼容总结

这是我总结多年的一个小文档，主要内容是 Javascript 和 CSS 浏览器兼容总结，最近看见有人咨询浏览器兼容的问题，就贡献出来。

并不一定全，有的也可能不准确，比如新出的 IE8、Chrome 等都没有太多涉及，虽然最近做的一些项目也兼容了 IE8、Chrome 等，但都没来的及总结进去，后来就忘了…汗。大家一起慢慢完善吧。

javascript 部分

1. document.form.item 问题

问题：

代码中存在 document.formName.item(“itemName”) 这样的语句，不能在 FF 下运行

解决方法：

改用 document.formName.elements["elementName"]

2. 集合类对象问题

问题：

代码中许多集合类对象取用时使用()，IE 能接受，FF 不能

解决方法：

改用 [] 作为下标运算，例：

document.getElementsByName(“inputName”)(1) 改为

document.getElementsByName(“inputName”)[1]

3. window.event

问题：

使用 window.event 无法在 FF 上运行

解决方法：

FF 的 event 只能在事件发生的现场使用，此问题暂无法解决。可以把 event 传到函数里变通解决：

```
onMouseMove = “functionName(event)”
function functionName (e) {
    e = e || window.event;
    ……
}
```

4. HTML 对象的 id 作为对象名的问题

问题：

在 IE 中，HTML 对象的 ID 可以作为 document 的下属对象变量名直接使用，在 FF 中不能

解决方法：

使用对象变量时全部用标准的 getElementById(“idName”)

5. 用 idName 字符串取得对象的问题

问题：

在 IE 中，利用 eval(“idName”) 可以取得 id 为 idName 的 HTML 对象，在 FF 中不能

解决方法：

用 getElementById(“idName”) 代替 eval(“idName”)

6. 变量名与某 HTML 对象 id 相同的问题

问题：

在 FF 中，因为对象 id 不作为 HTML 对象的名称，所以可以使用与 HTML 对象 id 相同的变量名，IE 中不能

解决方法：

在声明变量时，一律加上 var ，以避免歧义，这样在 IE 中亦可正常运行

最好不要取与 HTML 对象 id 相同的变量名，以减少错误

7. event.x 与 event.y 问题

问题：

在 IE 中，event 对象有 x,y 属性，FF 中没有

解决方法：

在 FF 中，与 event.x 等效的是 event.pageX ，但 event.pageX IE 中没有

故采用 event.clientX 代替 event.x ，在 IE 中也有这个变量

event.clientX 与 event.pageX 有微妙的差别，就是滚动条

要完全一样，可以这样：

mX = event.x ? event.x : event.pageX;

然后用 mX 代替 event.x

8. 关于 frame

问题：

在 IE 中可以用 window.testFrame 取得该 frame，FF 中不行

解决方法：

window.top.document.getElementById(“testFrame”).src = ‘xx.htm’

window.top.frameName.location = ‘xx.htm’

9. 取得元素的属性

在 FF 中，自己定义的属性必须 getAttribute() 取得

10. 在 FF 中没有 parentElement,parement.children 而用 parentNode, parentNode.childNodes

问题：

childNodes 的下标的含义在 IE 和 FF 中不同，FF 的 childNodes 中会插入空白文本节点

解决方法：

可以通过 node.getElementsByTagName() 来回避这个问题

问题：

当 html 中节点缺失时，IE 和 FF 对 parentNode 的解释不同，例如：

<form>

<table>

<input/>

</table>

</form>

FF 中 input.parentNode 的值为 form,而 IE 中 input.parentNode 的值为空节点

问题:

FF 中节点自己没有 removeNode 方法

解决方法:

必须使用如下方法 node.parentNode.removeChild(node)

11. const 问题

问题:

在 IE 中不能使用 const 关键字

解决方法:

以 var 代替

12. body 对象

FF 的 body 在 body 标签没有被浏览器完全读入之前就存在，而 IE 则必须在 body 完全被读入之后才存在

这会产生在 IE 下,文档没有载入完时,在 body 上 appendChild 会出现空白页面的问题

解决方法:

一切在 body 上插入节点的动作，全部在 onload 后进行

13. url encoding

问题:

一般 FF 无法识别 js 中的&

解决方法:

在 js 中如果书写 url 就直接写&不要写&

14. nodeName 和 tagName 问题

问题:

在 FF 中，所有节点均有 nodeName 值，但 textNode 没有 tagName 值，在 IE

中，nodeName 的使用有问题

解决方法：

使用 tagName，但应检测其是否为空

15. 元素属性

IE 下 input.type 属性为只读，但是 FF 下可以修改

16. document.getElementsByName() 和 document.all[name] 的问题

问题：

在 IE 中，getElementsByName()、document.all[name] 均不能用来取得 div 元素

是否还有其它不能取的元素还不知道(这个问题还有争议，还在研究中)

17. 调用子框架或者其它框架中的元素的问题

框架问题繁体字网的前段设计师曾做过详细的讲解，简单来说，在 IE 中，可以用如下方法来取得子元素中的值

document.getElementById(“frameName”).(document.)elementName

window.frames["frameName"].elementName

在 FF 中则需要改成如下形式来执行，与 IE 兼容：

window.frames["frameName"].contentWindow.document.elementName

window.frames["frameName"].document.elementName

18. 对象宽高赋值问题

问题：

FireFox 中类似 obj.style.height = imgObj.height 的语句无效

解决方法：

统一使用 obj.style.height = imgObj.height + “px”;

19. innerText 的问题

问题：

innerText 在 IE 中能正常工作，但是 innerText 在 FireFox 中却不行

解决方法：

在非 IE 浏览器中使用 textContent 代替 innerText

20. event.srcElement 和 event.toElement 问题

问题：

IE 下，even 对象有 srcElement 属性，但是没有 target 属性；Firefox 下，even 对象有 target 属性，但是没有 srcElement 属性

解决方法：

```
var source = e.target || e.srcElement;
var target = e.relatedTarget || e.toElement;
```

21. 禁止选取网页内容

问题：

FF 需要用 CSS 禁止，IE 用 JS 禁止

解决方法：

```
IE: obj.onselectstart = function() {return false;}
FF: -moz-user-select:none;
```

22. 捕获事件

问题：

FF 没有 setCapture()、releaseCapture()方法

解决方法：

```
IE:
obj.setCapture();
obj.releaseCapture();
FF:
window.captureEvents(Event.MOUSEMOVE|Event.MOUSEUP);
window.releaseEvents(Event.MOUSEMOVE|Event.MOUSEUP);
if (!window.captureEvents) {
        o.setCapture();
}else {
        window.captureEvents(Event.MOUSEMOVE|Event.MOUSEUP);
}
if (!window.captureEvents) {
        o.releaseCapture();
```

```
}else {
        window.releaseEvents(Event.MOUSEMOVE|Event.MOUSEUP);
}
```

CSS 部分

div 类

1. 居中问题

div 里的内容，IE 默认为居中，而 FF 默认为左对齐

可以尝试增加代码 margin:auto

2. 高度问题

两上下排列或嵌套的 div，上面的 div 设置高度(height)，如果 div 里的实际内容大于所设高度，在 FF 中会出现两个 div 重叠的现象；但在 IE 中，下面的 div 会自动给上面的 div 让出空间

所以为避免出现层的重叠，高度一定要控制恰当，或者干脆不写高度，让他自动调节，比较好的方法是 height:100%;

但当这个 div 里面一级的元素都 float 了的时候，则需要在 div 块的最后，闭和前加一个沉底的空 div，对应 CSS 是：

```
.float_bottom
{clear:both;height:0px;font-size:0px;padding:0;margin:0;border:0;line
-height:0px;overflow:hidden;}
```

3. clear:both;

不想受到 float 浮动的，就在 div 中写入 clear:both;

4. IE 浮动 margin 产生的双倍距离

```
#box {
float:left;
width:100px;
margin:0 0 0 100px; //这种情况之下 IE 会产生200px 的距离
display:inline; //使浮动忽略
}
```

5. padding 问题

FF 设置 padding 后，div 会增加 height 和 width，但 IE 不会（* 标准的 XHTML1.0 定义 dtd 好像一致了）

高度控制恰当，或尝试使用 height:100%;

宽度减少使用 padding

但根据实际经验，一般 FF 和 IE 的 padding 不会有太大区别，div 的实际宽 = width + padding ，所以 div 写全 width 和 padding，width 用实际想要的宽减去 padding 定义

6. div 嵌套时 y 轴上 padding 和 marign 的问题

FF 里 y 轴上 子 div 到 父 div 的距离为 父 padding + 子 marign

IE 里 y 轴上 子 div 到 父 div 的距离为 父 padding 和 子 marign 里大的一个

FF 里 y 轴上 父 padding=0 且 border=0 时，子 div 到 父 div 的距离为0，子 marign 作用到 父 div 外面

7. padding，marign，height，width 的傻瓜式解决技巧

注意是技巧，不是方法：

写好标准头

<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”

“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>

<html xmlns=”http://www.w3.org/1999/xhtml”>

高尽量用 padding，慎用 margin，height 尽量补上100%，父级 height 有定值子级 height 不用100%，子级全为浮动时底部补个空 clear:both 的 div

宽尽量用 margin，慎用 padding，width 算准实际要的减去 padding

列表类

1. ul 标签在 FF 中默认是有 padding 值的，而在 IE 中只有 margin 有值

先定义 ul {margin:0;padding:0;}

2. ul 和 ol 列表缩进问题

消除 ul、ol 等列表的缩进时，样式应写成:

{list-style:none;margin:0px;padding:0px;}

显示类

1. display:block,inline 两个元素

display:block; //可以为内嵌元素模拟为块元素

display:inline; //实现同一行排列的的效果

display:table; //for FF,模拟 table 的效果

display:block 块元素，元素的特点是：

总是在新行上开始；

高度，行高以及顶和底边距都可控制；

宽度缺省是它的容器的100%，除非设定一个宽度

，<p>，<h1>，<form>， 和 <li> 是块元素的例子

display:inline 就是将元素显示为行内元素，元素的特点是：

和其他元素都在一行上；

高，行高及顶和底边距不可改变；

宽度就是它的文字或图片的宽度，不可改变。

，<a>，<label>，<input>，<img>， 和 是 inline 元素的例子

2. 鼠标手指状显示

全部用标准的写法 cursor: pointer;

背景、图片类

1. background 显示问题

全部注意补齐 width，height 属性

2. 背景透明问题

IE: filter: progid:

DXImageTransform.Microsoft.Alpha(style=0,opacity=60);

IE: filter: alpha(opacity=10);

FF: opacity:0.6;

FF: -moz-opacity:0.10;

最好两个都写，并将 opacity 属性放在下面

原文链接：http://blogread.cn/it/article/3977?f=hot1&utm_source=tuicool

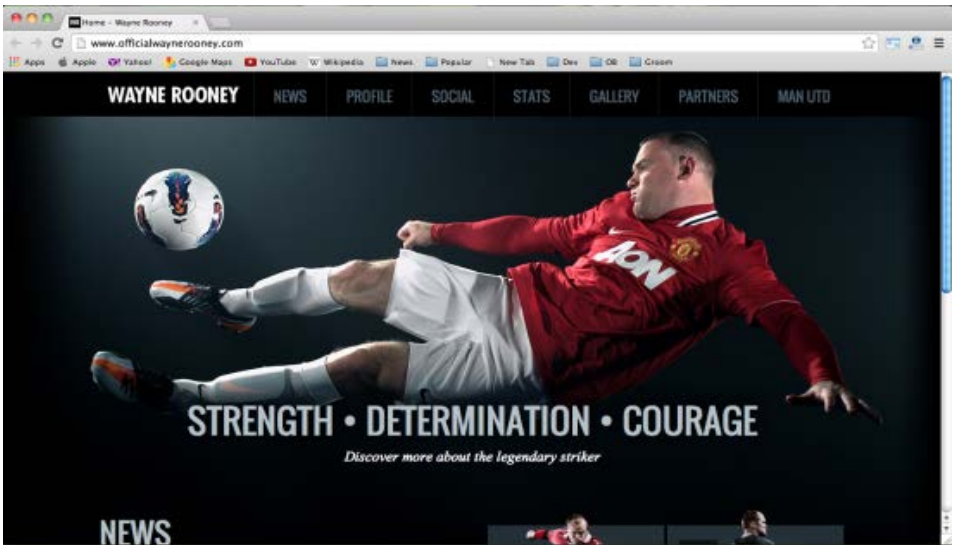
盘点 2013：最优秀的 HTML5&CSS3 设计

Web 设计是一个活到老、学到老的领域，每年都会有一些新的流行元素或者新技术出现，为了跟随潮流，Web 设计/开发人员要不停地给自己充电。坦白讲，Web 开发并没有太多的规则或约束，设计/开发人员可以充分发挥自己的创新能力和长处。开发出一些新的、有创意、受欢迎的作品一直是大家所追寻的目标。毫无疑问，2013年，Web 设计领域的新技术 HTML5&CSS3的快速发展旨在创造出更多更优秀的作品出来。尽管它们只是版本上的更新，但对于许多人来说，它们仍然是个新鲜的东西。不断出现新东西并且学习它们，把这些新功能添加到相应的系统中，这或许也是 Web 设计/开发的乐趣所在。但无论如何，你都不能喜新厌旧，一些优秀的老设计模式是永远都不能丢弃的。

因此，为了帮助设计师和开发人员快速的掌握和运用新技术，一些专家会专门推出一些教程，这些教程不仅加深你对新技术的理解，同时也帮助你基于原有的技术掌握更多的开发技能，开发出更多优秀的作品。作者搜集了2013年基于HTML5&CSS3开发的一些优秀作品，希望这些作品能给你带来更多的开发设计灵感。

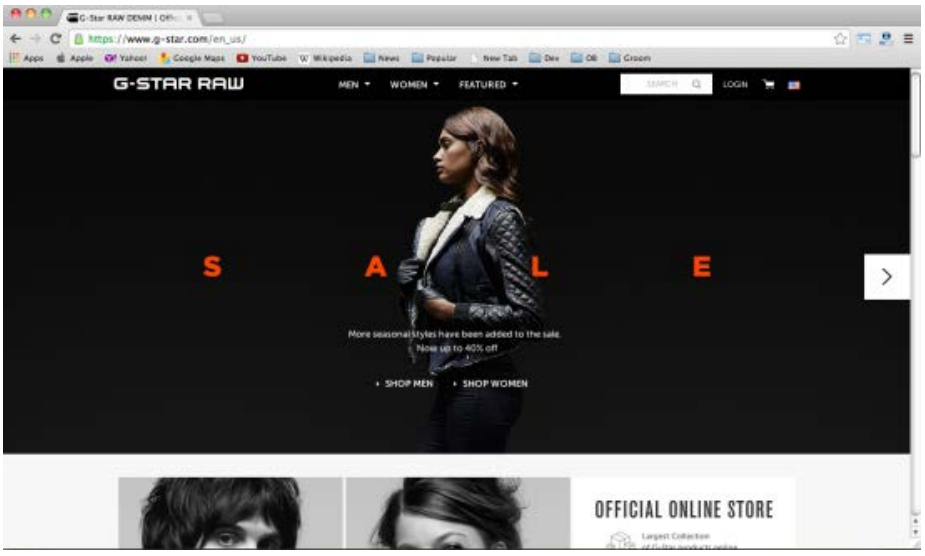
1. Une Cuisine

2. Official Wayne Rooney



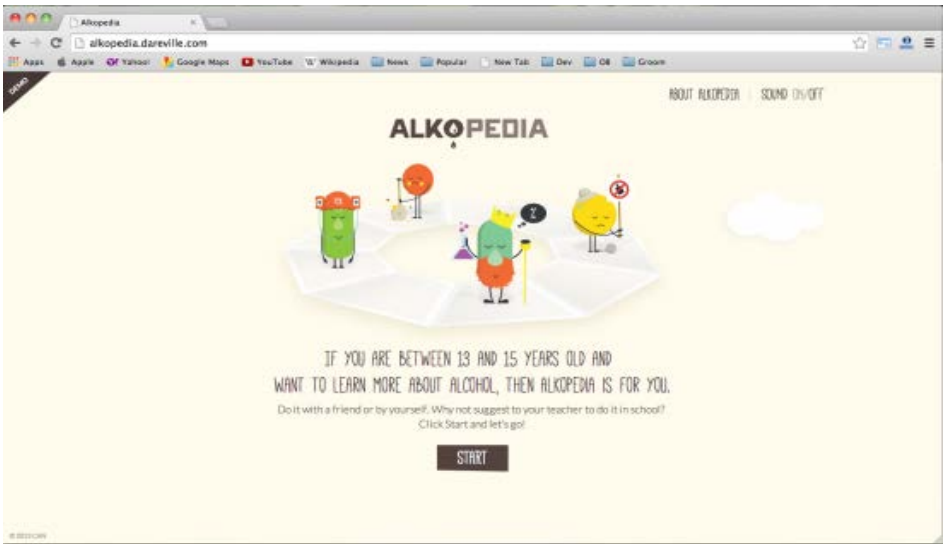
3. New Balance

4. G Star



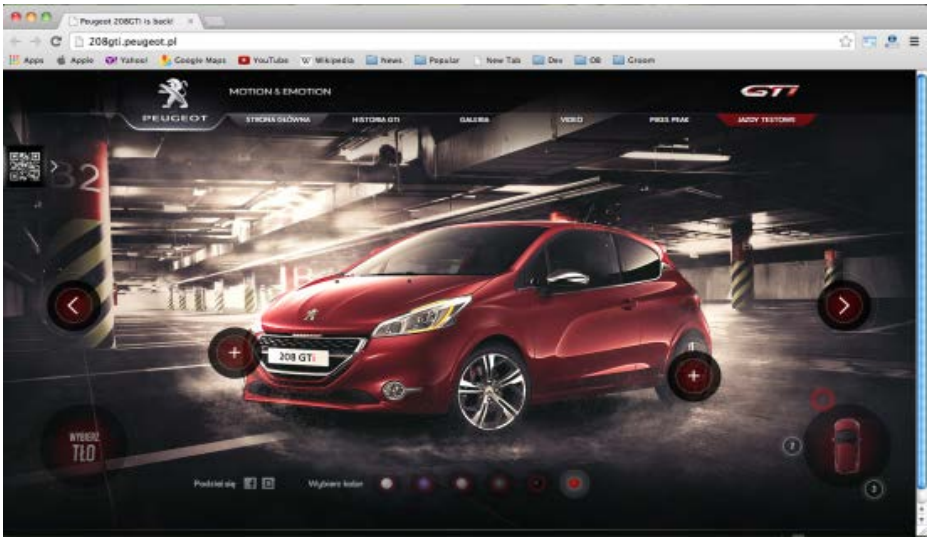
5. CSS Piffle

6. [Alko Pedia]

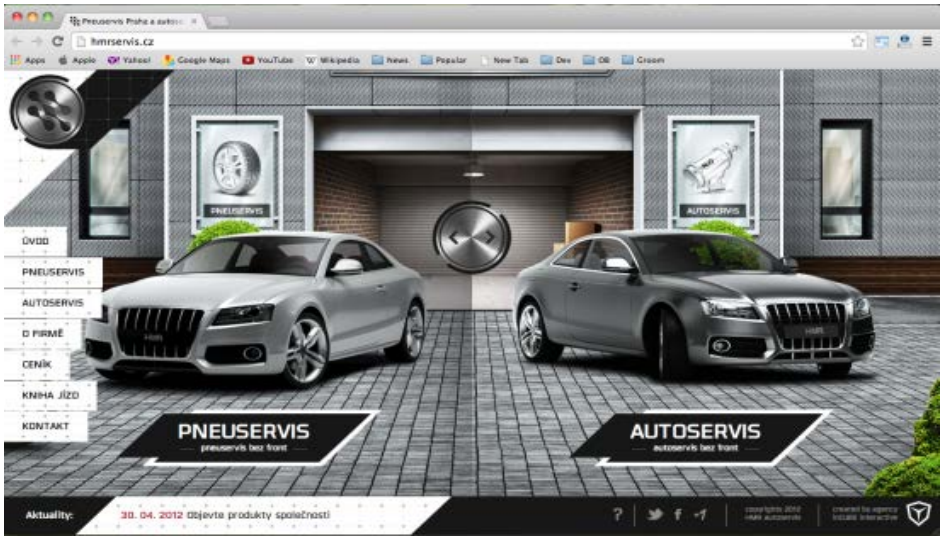


7. [Jobs is Free]

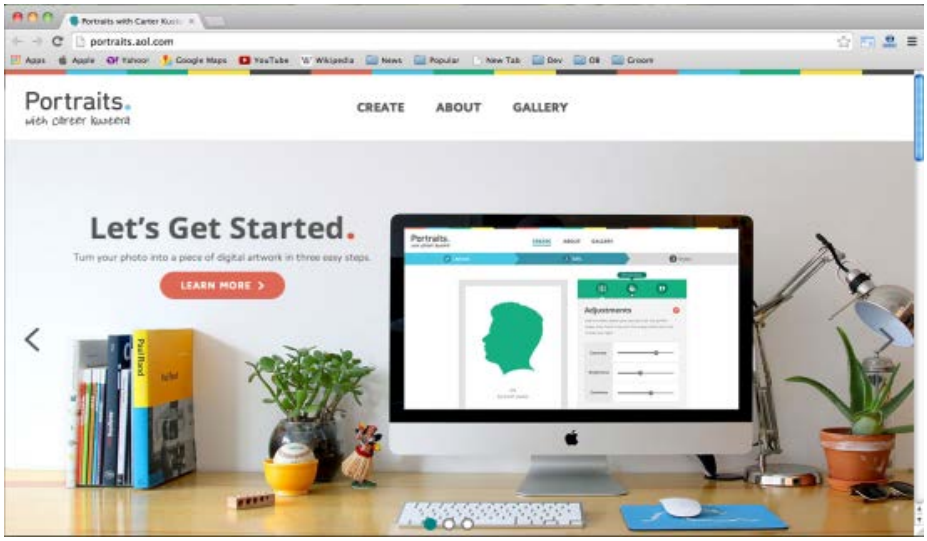
8. [Peugeot]



9. [Hmrservis]

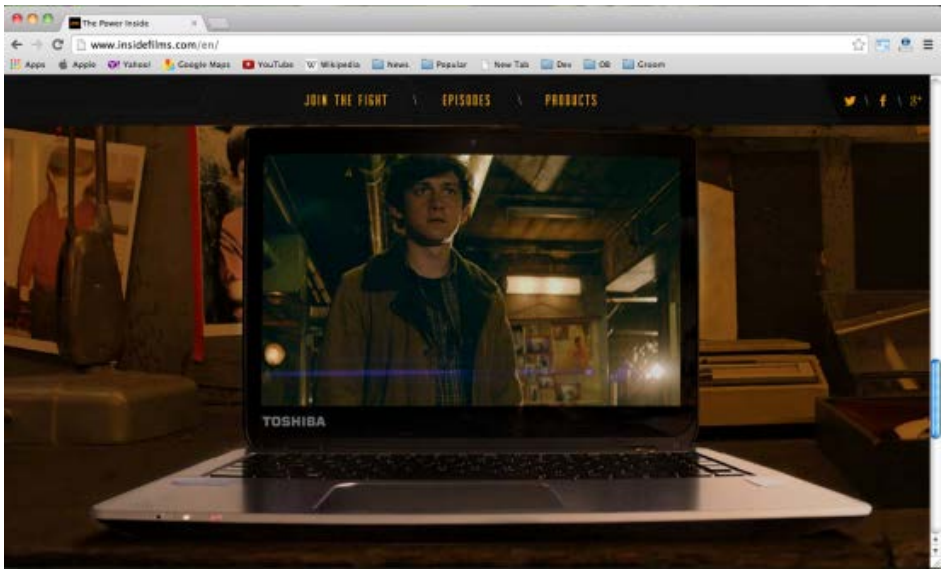


10. [Portraits]

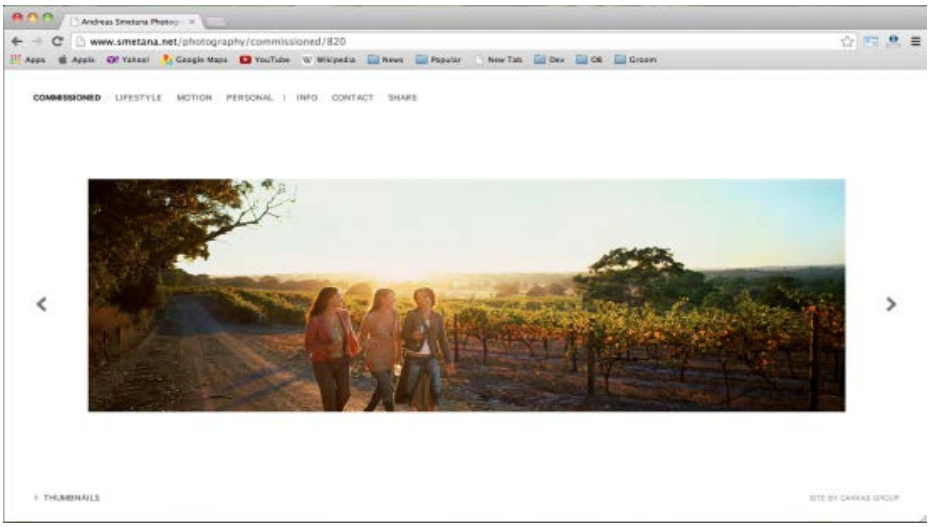


11. [Soleilnoir]

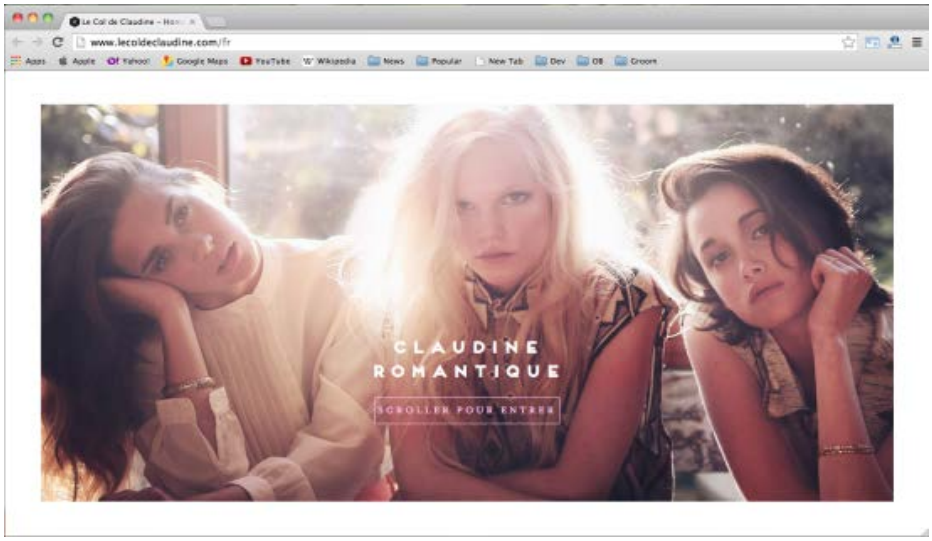
12. [Inside Films]



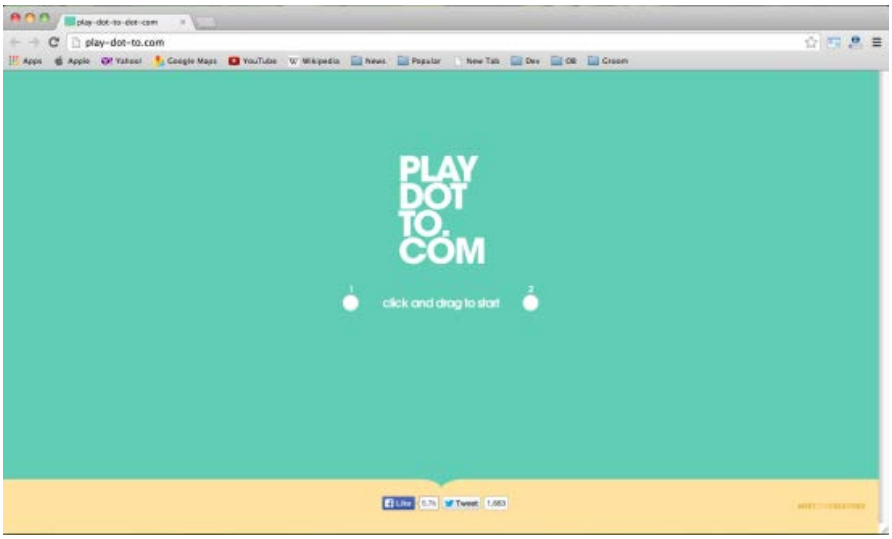
13. [Smetana]



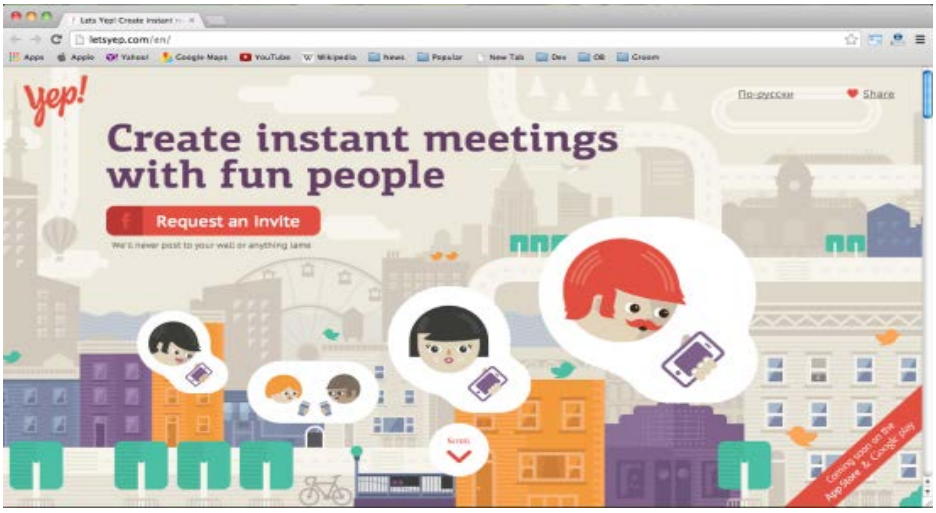
14. [lecoldeclaudine]



15. [Play dot to]

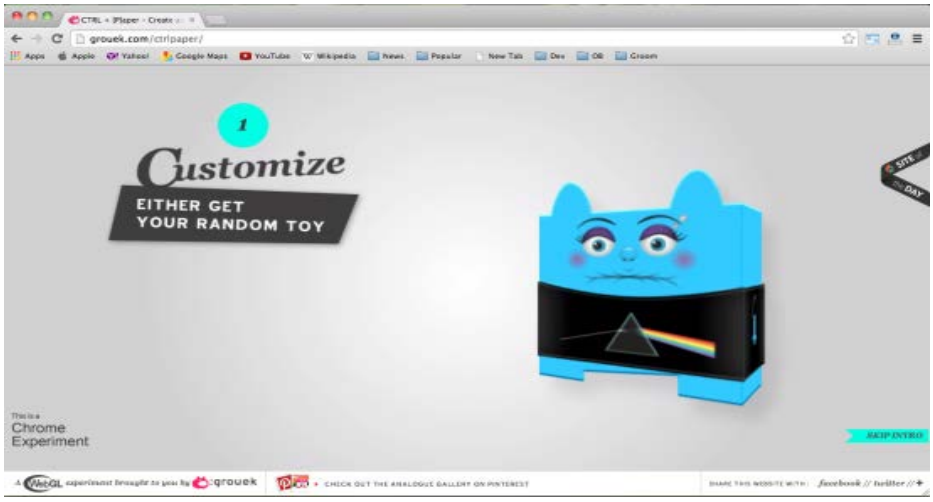


16. [Lets Yep]

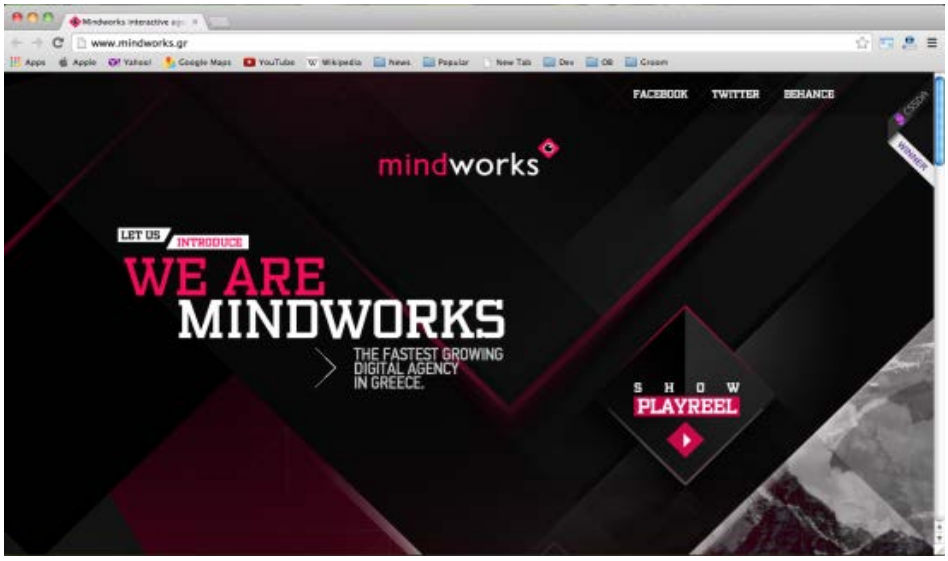


17. [The Penthouse Project]

18. [Ctrl Paper]

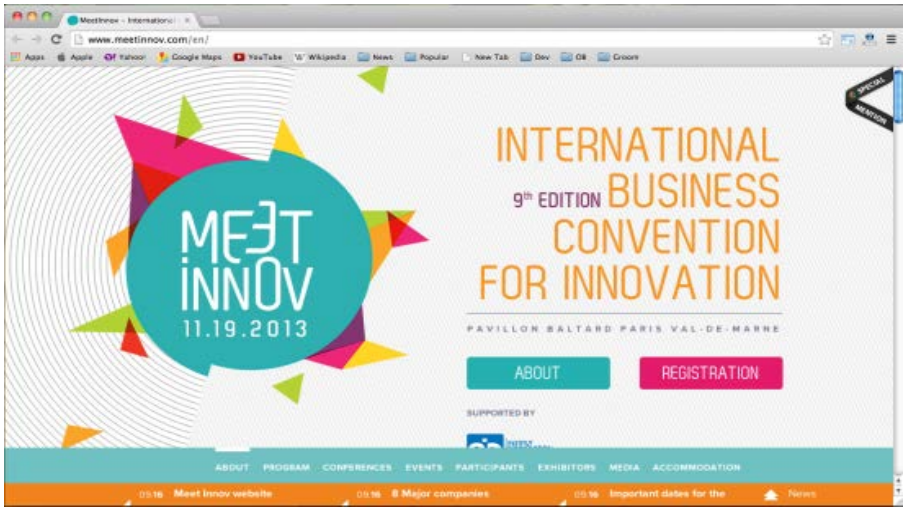


19. [Mind Works]

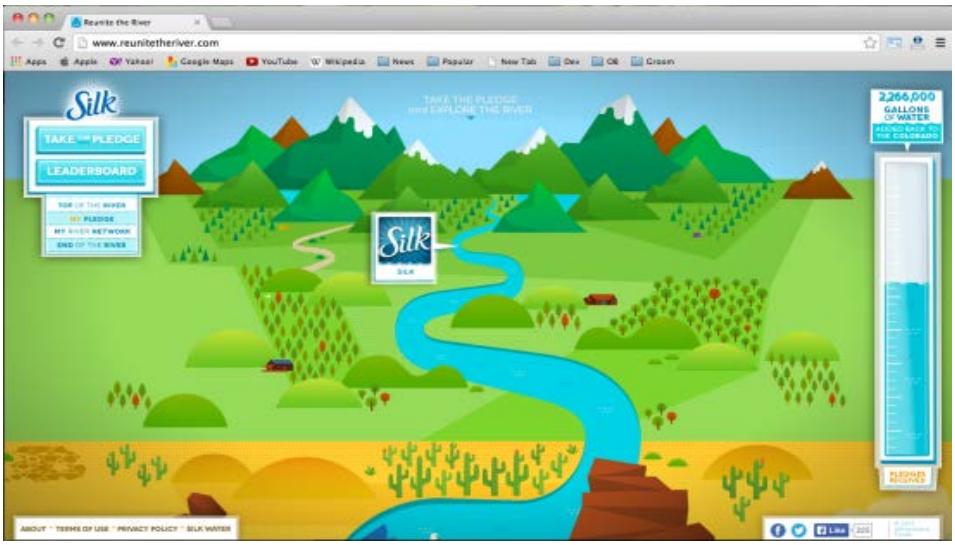


20. [Suitupordie]

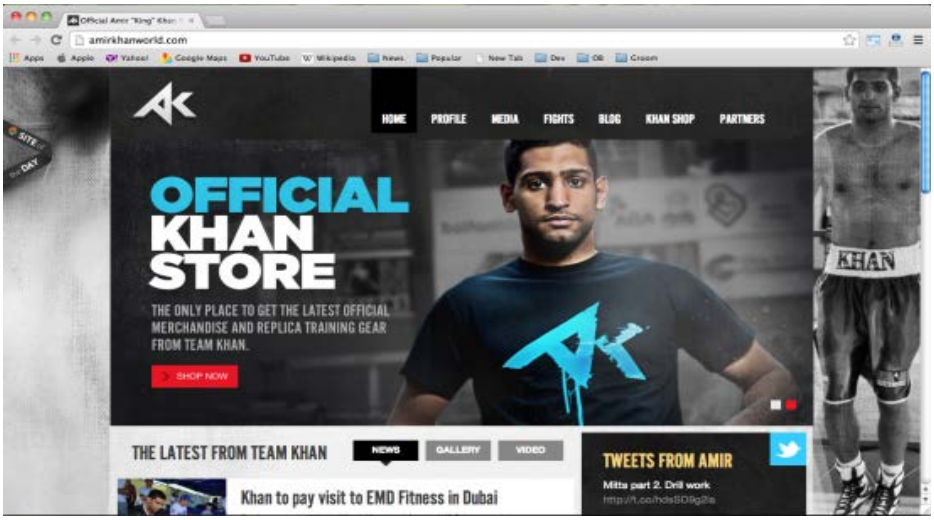
21. [Meetinnov]



22. [Reunite the River]

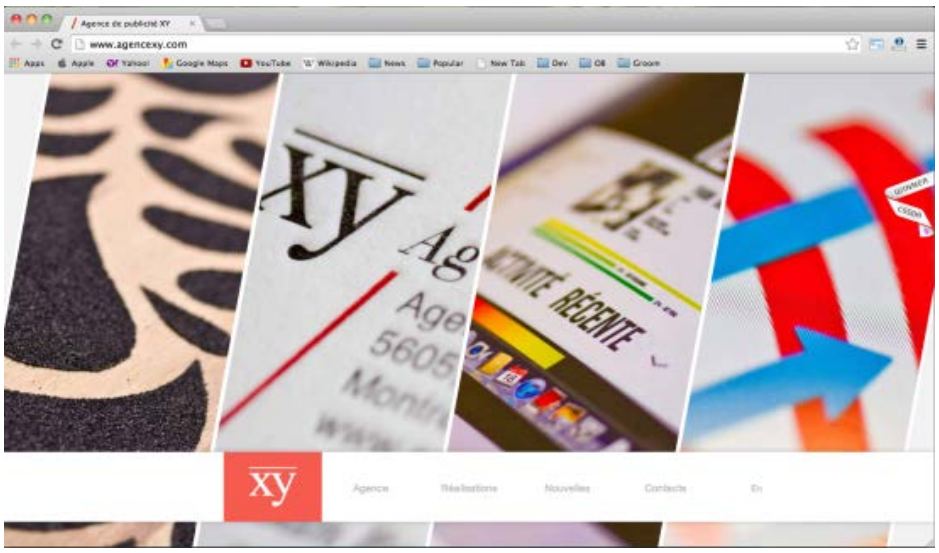


23. [Amir khan World]

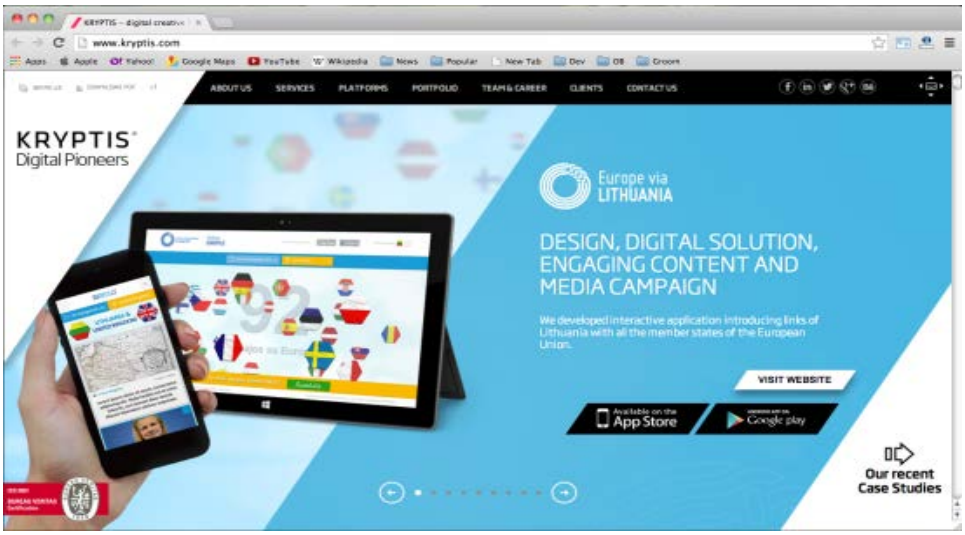


24. [Diadobaralho]

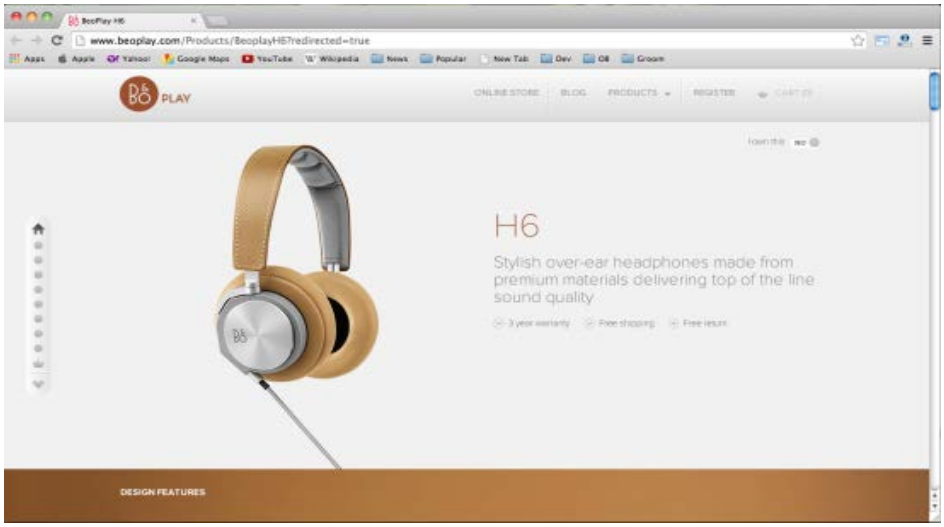
25. [Agencexy]



26. [Kryptis]

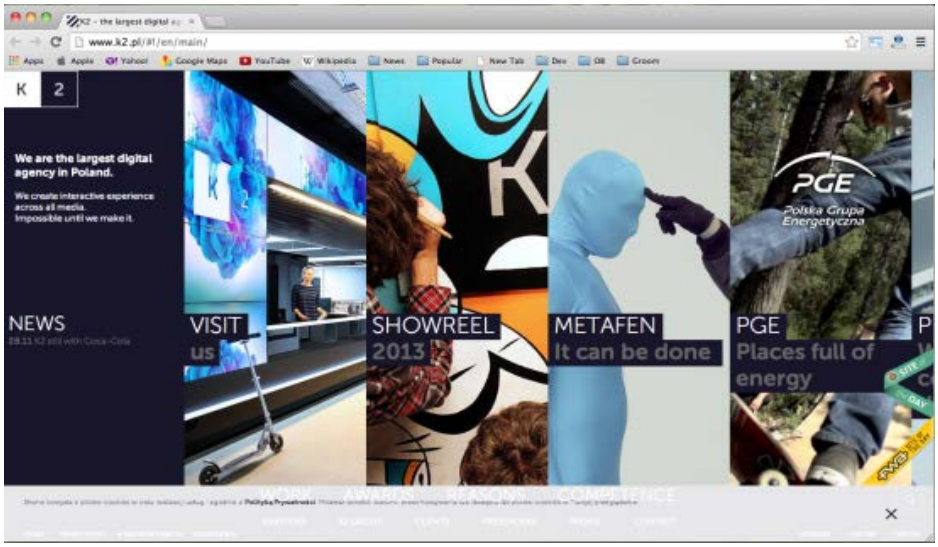


27. [Beoplay]

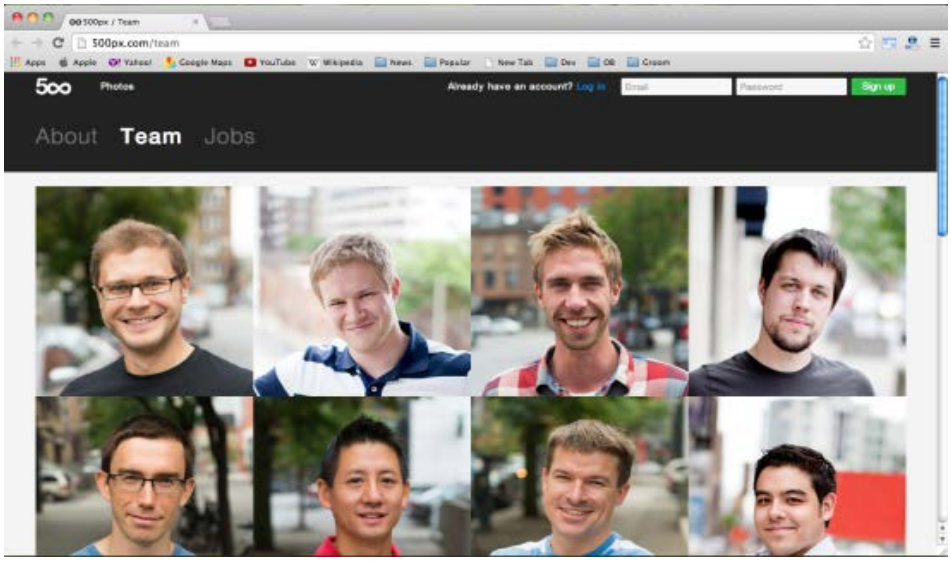


28. [Cheese Please Game]

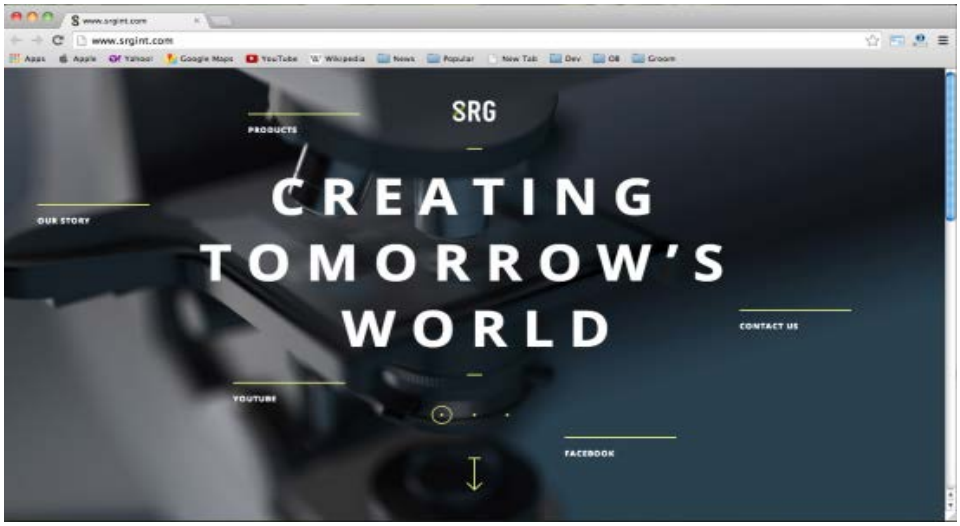
29. [K2]



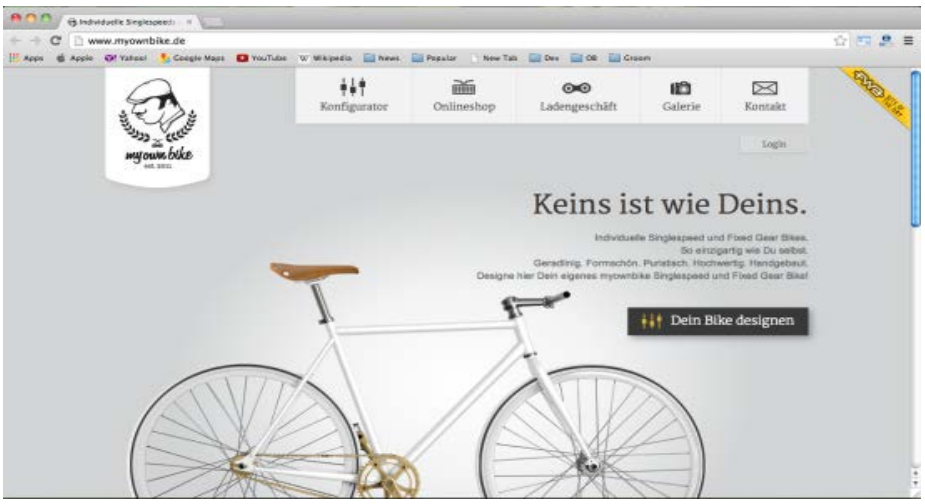
30. [500]



31. [Srgint]



32. [My Own Bike]



33. [Angry Birds Space]

原文链接：

http://www.csdn.net/article/2014-01-16/2818160-the-best-list-of-html5-css3-web-designs-of-2013?utm_source=tuicool

浏览器中关于事件的那点事儿

在前端中，有一个很重要的概念就是事件. 我对于事件的理解就是使用者对浏览器进行的一个动作，或者说一个操作.

本文会介绍很多与事件有关的东西，虽然我的出发点有那么点一网打尽的意思，不过也难以盖全. 所以就把最常用，最基本也相对重要的内容拿出来记录一下.

Javascript 绑定事件的方式

传统的事件绑定

因为各种历史原因，事件的绑定在不同的浏览器总是有不同的写法，当然现在可能大多数人都已经习惯于 jQuery 的事件绑定，而不清楚 javascript 的原生事件绑定是什么样子.

非常传统的事件的绑定方式，是在一个元素上直接绑定方法，element.onclick = function(e){}

```
<body>
  <input type="button" id="bt" name="bt button" value="this is a button">
  <script>
    var bt = document.getElementById("bt");
    bt.onclick = function(e){
      alert("this is a alert");
      alert(e.currentTarget.name);
    }
  </script>
</body>
```

这是传统的事件绑定，它非常简单而且稳定，适应不同浏览器. e 表示事件，this 指向当前元素. 但是这样的绑定只会在事件冒泡中运行，捕获不行. 一个元素一次只能绑定一个事件函数.

W3C 方式的事件绑定

W3C 中推荐使用的事件绑定是用 addEventListener()函数，element.addEventListener('click',function(e){...},false)，上代码:

```
<body>
  <input type="button" id="bt" name="bt button" value="this is a button">
  <script>
    var bt = document.getElementById("bt");
    bt.addEventListener('click',function(e){
      alert("this is a alert");
      alert(e.currentTarget.name);
    },false);
  </script>
</body>
```

如此的效果和之前的传统绑定方式是一样的，这种绑定同时支持捕获和冒泡，addEventListener()函数最后的函数表达了事件处理的阶段，false(冒泡)，true(捕获)。这种方式最重要的好处就是对同一元素的同一个类型事件做绑定不会覆盖，会全部生效。比如对上面代码bt元素在进行一次click的绑定，那么两次绑定的事件处理函数都会执行，按照代码书写顺序。

但是IE浏览器不支持addEventListener()函数，只在IE9以上(包括IE9)可以使用。IE浏览器相应的要使用attachEvent()函数代替。

IE下的事件绑定

IE下事件绑定的函数是attachEvent，它支持全系列的IE。但是如果你在Chrome等其他内核浏览器中使用这个函数去绑定事件，浏览器会报错的。

```
<body>
  <input type="button" id="bt" name="bt button" value="this is a button">
  <script>
    var name = "world";
    var bt = document.getElementById("bt");
    bt.attachEvent('onclick',function(){
      alert("hello "+ this.name);
    });
  </script>
</body>
```

attachEvent()函数支持事件捕获的冒泡阶段，同时它不会覆盖事件的绑定。但是一个缺点就是它处理函数中的this指向的是全局的window，所以上面代码弹出的结果会是"hello world"。

冒泡和捕获

上面的绑定事件中提到了冒泡和捕获阶段的概念，这两个概念对于理解事件是很重要的，对于它们的理解还要涉及到DOM(文档对象模型)和事件流的概念。事件流就是一个事件对象沿着特定数据结构传播的这么一个过程。

所谓的事件对象就是Event，当一个元素上绑定的事件被触发时会产生一个事件对象，从一切皆对象的观点看这是很符合逻辑的。冒泡和捕获讲的就是事件流在DOM中两种不同的传播方式。对于冒泡和捕获的理解，我们还是从一个小的示例来看：

```
<body>
  <div id="bt1" style="width:300px;height:300px;border:1px solid red" name="divbt1">
    <div id="bt2" style="width:100px;height:100px;border:1px solid red" name="divbt2"></div>
  </div>
  <script>
    var bt1 = document.getElementById("bt1");
    bt1.onclick = function(e){
      alert("bt1");
    }
    var bt2 = document.getElementById("bt2");
    bt2.onclick = function(e){
      alert("bt2");
    }
  </script>
</body>
```

这里我们使用最简单的,最原始的事件绑定方式.2个 div 嵌套并且绑定有弹窗事件,那么当我们点击里面的div的时候,两个div的点击事件都会被触发这个是没有疑问的,那么它们的处理函数谁先被执行?

这里用 IE8,9,10和 Chrome 浏览器同时实验,结构都是先弹出 bt2,然后弹出 bt1.也就是里面小 div 的事件先被处理了.我们来思考一下这是什么样的一个顺序,从 DOM 的结构上看,应该是这样的 body > bt1 > bt2.我们把这个结构竖过来,bt2在整个结构的最下面,body 在最上面.想象一下,当点击发生时产生一个泡泡(也就是事件对象),然后这个泡泡慢慢向上浮,首先路过 bt2,然后路过 bt1,在路过它们时依次执行事件函数,这就是冒泡型事件.

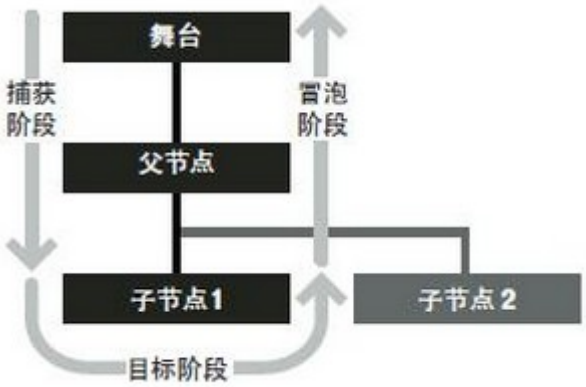
与之相反的就是捕获型事件,它事件流传播的顺序正好与冒泡型事件完全相反.也就是 bt1上的事件先触发,然后传递到 bt2.捕获是由表及里,冒泡是由内之外.

那么现在回忆一下之前的 W3C 标准中那个 addEventListener()函数,它里面最后一个参数 false 代表冒泡,true 代表捕获,这是什么意思呢?因为 W3C 作为一个标准,它选择了一个相对折中的方案去处理事件,也就是任何在 W3C 事件模型中发生的事件都先进入捕获阶段,然后在进入冒泡阶段,保证一个事件会经历这两个阶段,以适应 IE 和其他非 IE 浏览器,这样使用者可以根据需求选择将事件注册到哪一个阶段.

现在再来看用 addEventListener()函数进行事件绑定的结果:

```
<body>
  <div id="bt1" style="width:300px;height:300px;border:1px solid red" name="divbt1">
    <div id="bt2" style="width:100px;height:100px;border:1px solid red" name="divbt2"></div>
  </div>
  <script>
    var bt1 = document.getElementById("bt1");
    bt1.addEventListener('click',function(e){
      alert("bt1");
    },false);
    var bt2 = document.getElementById("bt2");
    bt2.addEventListener('click',function(e){
      alert("bt2");
    },false);
  </script>
</body>
```

这里2个 div 的事件绑定类型一共有4个可能的组合,2个 false;2个 true;1个 false,1个 true;1个 true,1个 false.这里分别试验下吧,记住按照 W3C 标准,捕获阶段会在冒泡之前.



jQuery 绑定事件的方式

上面我们记录了关于 javascript 原生的事件绑定的一些写法,这里我们在介绍一下通过 jQuery 进行事件绑定的方式. 首先来夸一夸 jQuery 的好,通过 jQuery 绑定让我们省去了考虑浏览器兼容和事件流程序的相关细节内容. jQuery 中对于事件的绑定称为委托,这是一个很好的定义,所谓委托,顾名思义就是自己不去做,我让别人帮我做这个事. jQuery 就是这么做的,让我们详细了解下.

.bind()

我们直接看代码,bind()函数使用很简单.

```
<body>
  <div id="div1" style="width:300px;height:300px;border:1px solid red" name="divbt1">
  <script>
    $("#div1").bind('click',function(e){
      alert("div1 " + e.currentTarget.name);
    });
  </script>
</body>
```

代码在 IE8,IE11,Chrome 运行都没有问题,我们简单翻译一下就是首先找到 id 为 div1的 div 对象,然后给这个对象绑定一个 click 事件. 现在来分析一下 bind(),首先如果用它绑定事件要有一个寻找 jQuery 对象的过程,其次如果要为大量的元素绑定事件那么要寻找大量的对象不说,每一个对象还要占用内存来存储相应的处理函数. 并且 bind()只能为当前已存在的 DOM 节点绑定事件,如果节点还没有产生 bind 是没有办法的.

所以说 bind()推荐在使用比较简单的情况中,绑定不多的节点并且没有新节点产生的情况. 如果比较复杂就推荐使用 delegate().

.delegate()

在 jQuery 中还有一个 live()函数也能处理类似的问题，但是不如 delegate()好用，所以这里就不介绍了. delegate()是为了突破单一 bind()方法的局限性，实现事件的委托. 我们先看代码来理解：

```
<body>
  <div id="div1" style="width:300px;height:300px;border:1px solid red" name="divbt1">
    <div id="div2" style="width:100px;height:100px;border:1px solid red" name="divbt2"></div>
  </div>
  <script>
    $("body").delegate('#div1','click',function(e){
      alert("div1");
    });
    $("body").delegate('#div2','click',function(e){
      alert("div2");
    });
  </script>
</body>
```

解读一下 delegate()函数，我们寻找到 body 标签的对象并调用 delegate()，这是把事件的执行委托给 body. 也就是监听整个 DOM 树，当触发事件的 DOM 节点的是 id 为 div1，触发触发事件的类型是 click 时，在事件传播到 body 时，我们执行相应的处理函数. body 怎么能知道这么多，它如何知道绑定在它身上的执行函数什么时候执行?jQuery 这些事件委托的原理根据事件冒泡的机制，广播的时候所有的节点都会知道，到底发生了什么!

DOM 在为页面中的每个元素分派事件时，相应的元素一般都在事件冒泡阶段处理事件. 在类似 body > div > a 这样的结构中，如果单击 a 元素，click 事件会从 a 一直冒泡到 div 和 body(即 document 对象). 因此，发生在 a 上面的单击事件，div 和 body 元素同样可以处理. 而利用事件传播(这里是冒泡)这个机制，就可以实现事件委托. 具体来说，事件委托就是事件目标自身不处理事件，而是把处理任务委托给其父元素或者祖先元素，甚至根元素(document).

事件的取消

在一些情况下我们需要阻止事件流的传播，或者解除之前绑定的事件. 在实际工作中经常会遇到类似的需求，尤其是事件流的阻止.

事件流阻止

某些事件的对象是可以取消的，这意味着可以阻止默认动作的发生. 事件对象是否可以取消，要通过 Event.cancelable 属性表示. 事件监听器可以调用 Event.preventDefault()取消事件对象的默认动作. Event.stopPropagation()方法可以阻止事件向上冒泡.

事件的阻止根据场景不同和浏览器不同有不同的处理，因为事件处理模型不

同的关系,如果在 IE 下 Event.returnValue = false 就可以.如果是非 IE 下,用 Event.preventDefault()阻止.事件流阻止,这里面阻止的是它的继续传播以及有可能产生的默认动作.这里举一个常见且简单的例子,就是 submit 类型按钮的点击.

```
<body>
  <form action="asd.action">
    <input type="submit" id="tijiao" value="submit"/>
  </form>
  <script>
    $("body").delegate('#tijiao','click',function(e){
      e.preventDefault();
    });
  </script>
</body>
```

这里点击按钮,form 表单默认的提交被阻止了,也就是其默认动作终止了.这里有一个强调的就是滚动事件.滚动也是经常遇到需要处理的事件类型,但是滚动的阻止有点特例,它不支持在委托里进行阻止.

说到这里我们感觉 Event.preventDefault()和 Event.stopPropagation()都可以阻止事件,那么它们有什么区别?

前者是通知浏览器不要执行与事件相关联的默认动作,比如 submit 类型的按钮点击会提交.后者是停止事件流的继续冒泡,但是它对 IE8及以下 IE 浏览器支持不好.如果直接使用 return false 则表示终止处理函数.

事件函数的解除绑定

和事件的绑定其实是相对应的,如果需要接触事件的绑定,运行对应的函数就可以了.如果是原生 JS 绑定则对应运行 removeEventListener()和 detachEvent().看一个代码示例:

```
var EventUtil = {
  //注册
  addHandler: function(element, type, handler){
    if (element.addEventListener){
      element.addEventListener(type, handler, false);
    } else if (element.attachEvent){
      element.attachEvent("on" + type, handler);
    } else {
      element["on" + type] = handler;
    }
  },
  //移除注册
  removeHandler: function(element, type, handler){
    if (element.removeEventListener){
            element.removeEventListener(type, handler, false);
    } else if (element.detachEvent){
            element.detachEvent("on" + type, handler);
    } else {
            element["on" + type] = null;
    }
  }
};
```

如果是 jQuery 的绑定,也是存在对应的解绑函数用以清除注册事件,比如 unbind()和 undelegate().

原文:http://my.oschina.net/blogshi/blog/192658?utm_source=tuicool

了解 Json 和 XML

这是我编的第3部分“会议应用程序的秘密。 ”

在第1部分中，我演示了如何 dispay 一个最终用户许可协议在应用程序第一次启动。

第2部分是关于一个自定义凭据对话框。

这一次，我想讨论一下如何将数据永久保存，以保证快速响应的应用程序。有人问我为什么我的会议应用程序正在执行如此令人难以置信的快速而 retreiving 会议，演讲，或搜索（它得到正因为如此许多五星级评价） 。你应该下载 BBJam 亚洲会议应用程序，看看它有在行动。答案是不容易的，会涉及到一些更多的文章。让我先解释一下…

BlackBerry Jam Asia 使用的 JSON 数据模型

我使用 JSON 来坚持我的发布会上的应用程序的数据，并有很多原因。使用 REST 接口，你仍然可以从很多 XML 格式的响应的数据，并且在大多数情况下，你都需要消耗这些数据和不能改变的后端。第一件事情我一直做的是继续之前，将 XML 数据转换成 JSON 数据。你会发现这个示例应用程序中使用的所有数据文件和源代码在 GitHub 上。

我不能告诉你关于 BBJam 亚洲会议 API 的细节，所以我产生了一些样机数据：

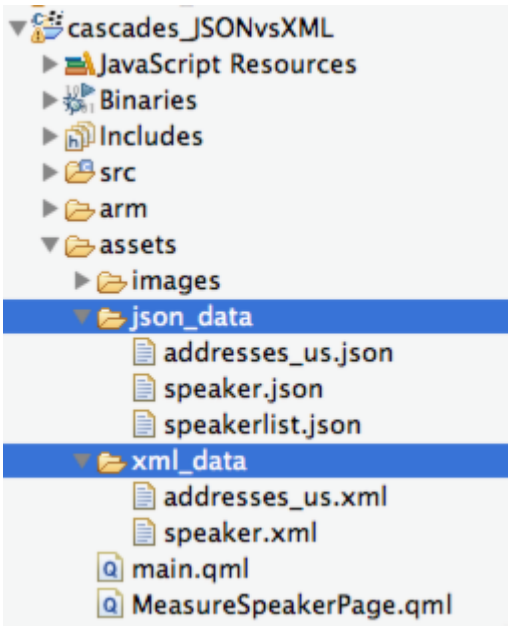
10,000随机地址有一个大的文件来试验

119市民提供的扬声器数据有一些会议相关的数据

从 fakenamegenerator.com 免费服务产生我使用的地址。如果你有提供样机的应用程序数据，或者如果你想测试使用大型数据集我强烈推荐这个服务给开发人员。他们甚至产生不同的数据为您选择的国家和地理坐标。

扬声器的数据是从另一个会议应用程序 WordPress 导出文件，包含来自公开网站的真实数据。地址和扬声器的数据是 XML 格式。我会告诉你如何将这些文件转换成 JSON ，并解释为什么你应该这样做。

这两个数据集可以在这里找到：



在开发和测试，你经常要看一看的数据，JSON 是更容易比 XML 来读取。下面是在 XML 的一个地址：

```
<Address>
  <City>Lincoln</City>
  <Company>Record Bar</Company>
  <Country>US</Country>
  <CountryFull>United States</CountryFull>
  <Domain>AffordableShow.com</Domain>
  <EmailAddress>SharonSMcCall@cuvox.de</EmailAddress>
  <GivenName>Sharon</GivenName>
  <Latitude>40.722909</Latitude>
  <Longitude>-96.678577</Longitude>
  <Number>1</Number>
  <State>NE</State>
  <StreetAddress>3002 Poling Farm Road</StreetAddress>
  <Surname>McCall</Surname>
  <TelephoneNumber>402-310-0424</TelephoneNumber>
  <Title>Mrs.</Title>
  <ZipCode>68501</ZipCode>
</Address>
```

这里，在同一个地址 JSON：

```
{
  "City" : "Lincoln",
  "Company" : "Record Bar",
  "Country" : "US",
  "CountryFull" : "United States",
  "Domain" : "AffordableShow.com",
  "EmailAddress" : "SharonSMcCall@cuvox.de",
  "GivenName" : "Sharon",
  "Latitude" : "40.722909",
  "Longitude" : "-96.678577",
  "Number" : "1",
  "State" : "NE",
  "StreetAddress" : "3002 Poling Farm Road",
  "Surname" : "McCall",
  "TelephoneNumber" : "402-310-0424",
  "Title" : "Mrs.",
  "ZipCode" : "68501"
}
```

因此，提高可读性是一个原因使用 JSON。如果你使用 QML，那么你可以很容易地从 JSON 对象复制并粘贴直接进入喜欢你的 QML 代码：

变种 myAddress={…. JSON 对象…} … myDataModel.insert(myAddress)….

有时候，这是您模拟对象插入到列表 DataModels 模拟数据非常有用。在大多数情况下的 JSON 数据文件较小然后 XML：

10,000 addresses in XML: 6.7 MB

10,000 addresses in JSON: 5.9 MB

如果 XML 是用许多长属性名的文件大小可以减少更多，但有时如果 XML 结构非常复杂，它可能是一个 JSON 文件越大则 XML（可看看扬声器的数据）。

也许你已经知道，瀑布支持 JSON 的非常好。例如，如果在 QML，其中 QVariantMap 被自动映射到 JavaScript 对象(== JSON)使用 QVariants 从 C++。也许对映的这个深度整合是原因….

大多数时候，里面一个会议应用程序，我从缓存中读取数据，这是在明确的 JSON 夺过来的 XML。下面是一些我值则采用10,000地址：

READ JSON: 2,392 ms

READ XML: 71,342 ms

规则＃1：如果缓存数据，请始终使用 JSON - 本可以更快高达30倍那么 XML

这是奇怪的：写数据到 JSON 是有点慢那么 XML。下面是我测量了10,000地址：

写的 JSON：5,255毫秒

编写 XML：3,567毫秒

写入速度差别是边际，你可以做异步写入，并继续你正在等待从读得到的数据，同时工作。

在某些情况下，你得到的比你需要更多的数据。来看看来自 WordPress 的 XML 格式导出为 RSS 扬声器的数据。这两种文件格式，XML 和 JSON，不容易阅读，并包含在一个结构复杂的数据太多，但很容易的解决方法。只要你从 HTTP 响应获取数据，将其转换并删除不需要的属性。这会给你更多的速度。下面是使用119音箱的测量：

Read XML: 378 ms

Read JSON: 76 ms

Read JSON v2: 23 ms

JSON V2只包含你需要在一个不太复杂的结构的属性。

规则＃2：如果越来越复杂的数据中删除未使用的属性，以获得更快的速度

这只是无聊的比较数字，让我们用瀑布展示的基于 JSON 的持久性或缓存的速度，看看它是多么容易的 XML 和 JSON 之间转换数据。

该示例应用程序只使用两个 QML 文件：

main.qml

MeasureSpeakerPage.qml

Main.qml 包含 NavigationPane 显示页面可视化的读取和写入10,000个地址的测量。

MeasureSpeakerPage.qml 将被推之上，以可视化读取 XML，JSON 和 JSON-以下的物业扬声器的数据测量。

让我们来看看 main.qml 第一。也有一些 attachedObjects：

SystemToast

ComponentDefinition

QTimer 的

我们正在使用的 SystemToast 作为一种简单的方法来让用户知道，如果一个进程被启动或完成。

而第一页是 main.qml 里面直接定义，该 MeasureSpeakerPage 只会在需求推动，如果用户启动一个动作。为了避免 UI 控件不必要的创造，我们只定义组件，

并推前上方动态创建它。

该 QTimer 的是作为一种变通方法有击中行动项目，并在 C 中调用的过程+ +之间有一个小的延迟。正如我们上面所看到的，它需要很长的时间来从 XML 中读取10,000个地址;在一分钟！在执行这项任务，我们要禁用操作的项目是什么事异步，需要更多毫秒。

如果不使用 QTimer 的会出现没有足够的时间来禁用操作项。为什么呢？因为我们正在做的事情你永远不应该做在实际应用中：执行在 UI 线程长时间运行的任务。虽然网络请求是异步操作开箱，读文件是不是和会阻塞 UI 。

此演示应用程序的目的是让你觉得长的执行时间

所以我使用了 QTimer 的解决方法。现在,禁用 ActionItems 如预期般运作。以下是我做的：

```
function processFinished(){
    activityIndicator.stop()
    navPane.running=false
    switch (myTimer.usecase){
        case 0:
            infoToast.body="10'000 Addresses processed.\nread JSON..."
            infoToast.button.label="OK"
            infoToast.icon="asset:///images/ca_done.png"
            infoToast.exec()
            return
        case 1:
            .....
            default:
              return
    }
}
onCreationCompleted:{
    app.conversionDone.connect(processFinished)
}
```

所有行动项目会在一个任务正在运行通过使用从 NavigationPane 属性被禁用。只要一个 ActionItem 被触发，该属性’运行’设置为’true’。然后，QTimer 的启动时使用100毫秒的延迟。该 QTimer 的开始之前，系统有足够的时间来禁用所有行动项目。因为我只想使用一个 QTimer 的对象，我添加了一个属性’用例“知道哪些任务从 C + +应该启动。

你怎么现在得到的信息，该任务已经完成了吗？我们正在使用 Qt 的信号和槽，C+ +所发出的信号，并在 QML 我们连接一个函数（插槽）到这个信号。连接

本身是在 onCreationCompleted{}完成。

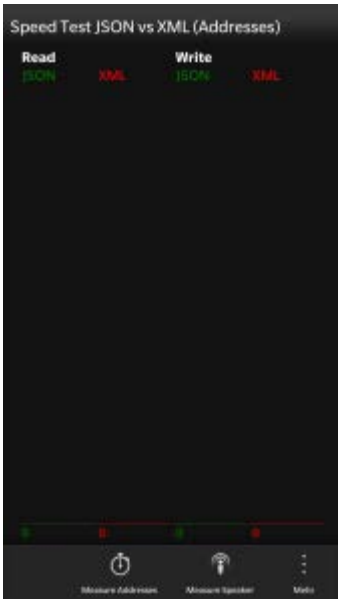
你会发现在连接的示例应用程序不同的老虎机（功能）。下面是呈现出吐司一个简单的函数，停止 ActivityIndicator 和属性’运行’设置为 false：

```
function processFinished(){
  activityIndicator.stop()
  navPane.running=false
  switch (myTimer.usecase){
    case 0:
      infoToast.body="10'000 Addresses processed.\nread JSON..."
      infoToast.button.label="OK"
      infoToast.icon="asset:///images/ca_done.png"
      infoToast.exec()
      return
    case 1:
      .....
      default:
        return
  }
}
onCreationCompleted:{
  app.conversionDone.connect(processFinished)
}
```

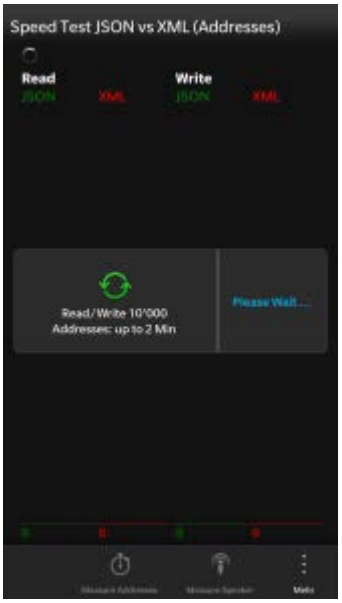
还有更复杂的插槽从 C + +得到的结果值，如果测量地址或扬声器。

下面是一些截图，看看从衡量地址结果的显示方式。

我们先从一个空的屏幕：



启动行动“测量地址”是指 ActionItems 将被禁用，举杯将显示，并且 ActivityIndicator 正在运行：



这是一个长时间运行的任务，可能需要长达两分钟！

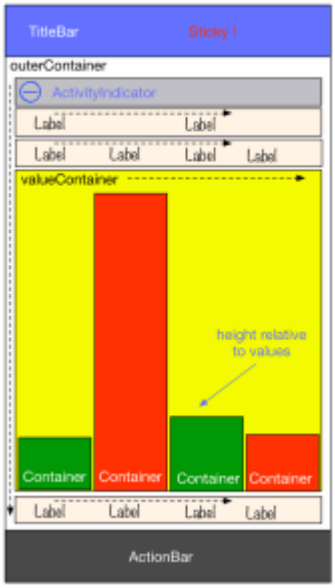
获得该任务完成的信号，吐司更改文本和图标，ActivityIndicator 停止，条形图绘制和 ActionItems 启用：

该 BarChart 的也应该在景观：

从这个 BarChart 的您轻松了解如何快速读取 JSON 文件，直接使用 XML 进行比较。你也看到，Z30比 Q10快得多（Z10是类似的）;81秒与117秒阅读10000地址的 XML 文件。如你所知，瀑布不提供 BarChart 的图表作为 uicontrol 的开箱即用，但它可以使用普通的容器来完成。

为了得到这一切工作顺利，也有一些小瀑布技巧，让我们来看看代码一步一步的。

我使用的 StackLayout，但你也可以使用一个 DockLayout - 它给你。对我来说，在这种情况下的 StackLayout 似乎更合适。以下是如何的容器是奠定了：



一切都包含在一个外部容器与 Top - >底方向：

该 ActivityIndicator ：唯一可见的和消费空间，如果运行使用的 StackLayout 左 - >右为使用 spaceQuota 调整它们是正确的标题标签两个容器使用的 StackLayout 左 - >右页脚标签一个容器在中间最困难的容器：在 valueContainer - 也使用包含4个货柜的 StackLayout 左 - >右：每个栏

该 valueContainer 要消耗所有可用的空间，所以尾标签总是被定位在底部。可用空间会改变，如果改变方向或启动和停止 ActivityIndicator 。

该 valueContainer 内的4箱代表的酒吧和充满背景颜色。酒吧的高度必须与我们从 C + + ，其中最大的值应填写完整的高度得到的值。所有其他的酒吧将获得相对于他们的价值的高度。这意味着我们必须重新计算这些容器的高度的高度的变化每一次。

把它做的伎俩是使用 LayoutHandlers 。一个 LayoutHandler 会在布局的变化得到通知，并可以要求处理有关的高度，这是我们所需要的值。我们附上 LayoutHandler 到 valueContainer ，如果高度变化，我们重新计算值。

```
Page {
    id: measureAddressesPage
    property int barHeight: 1200
    onBarHeightChanged: {
        calculateBarChart()
    }
    property int max: -1
    property int readJson: 0
    property int writeJson: 0
    property int readXml: 0
    property int writeXml: 0
    ......
    Container {
        id: valueContainer
        topPadding: 10
        preferredHeight: measureAddressesPage.barHeight
        layout: StackLayout {
            orientation: LayoutOrientation.LeftToRight
        }
        attachedObjects: [
            LayoutUpdateHandler {
                id: valueContainerLayoutHandler
                onLayoutFrameChanged: {
                    measureAddressesPage.barHeight = layoutFrame.height
                }
            }
        ]
        ......
    }
}
```

我们店里的值在页面属性（最大值， readJson ， writeJson ， ReadXml 的，writeXml ），并也有一个’ barHeight ‘属性。

在’ barHeight “属性绑定到 valueContainer 的 preferredHeight 并初步设定为高值：1200 。级联尝试将此值设置为 preferredHeight ，但它是智能只使用其装配在可用的空间的高度。正如我们已经讨论的那样，可用的空间可以改变。例如，在启动 ActivityIndicator 会降低高度，并停止将放大的高度。在这种情况下， LayoutHandler 会得到一个通知，我们将用实际分配的高度来重新计算值。

还有一些特殊的情况下，从 valueContainer 的 LayoutHandler 不会被通知有关布局的变化，但 outerContainer 将收到。因此，作为一个小窍门，我们也 LayoutHandler 添加到 outerContainer ，如果布局变化，我们简单地设置 valueContainer 的 preferredHeight 为高值（ 1200） ，造成瀑布抓住所有的可用空间等等。

另一种情况是回来从风景到人像，我们还设置此高度值，让瀑布做的可用空间的计算。

标题栏被设置为粘性。这是一个把戏，以获得正确的高度上的 Q10 。否则，

Q10将采用浮动标题栏，这意味着标题栏的高度将是“可用” ，然后如果滚动标题栏远离你的页脚标签将是唯一可见的。设置标题栏为粘解决这个问题。

请大家深入了解的示例代码：

业务逻辑（ C + +）

现在让我们看看在 C + +侧发生的事情。

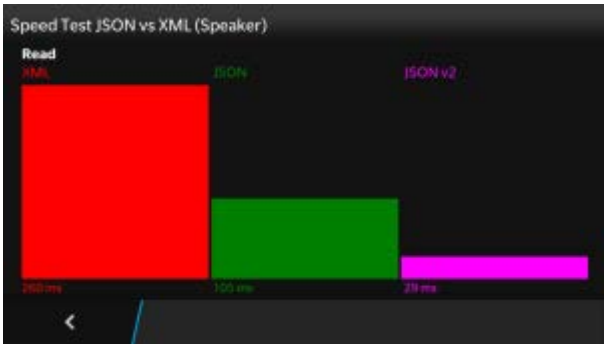
我们需要的“应用程序”上下文属性来访问应用程序。请记住，所有方法都必须从 QML 可调用。欲了解更多信息，看看源或本系列的前几篇文章。

我们现在要集中一个 JSON 和 XML 。您必须包括：

```
void ApplicationUI::convertXMLtoJSONAddresses() {
    XmlDataAccess xda;
    QVariant data;
    QString filenameXML;
    filenameXML = "addresses_us.xml";
    QFile dataFile(assetsXMLPath(filenameXML));
    if (!dataFile.exists()) {
        // do something
        return;
    }
    bool ok = dataFile.open(QIODevice::ReadOnly);
    if (ok) {
        data = xda.loadFromBuffer(dataFile.readAll());
        dataFile.close();
    } else {
        // do something
    }
    QString filenameJSON;
    filenameJSON = "addresses_us.json";
    JsonDataAccess jda;
    jda.save(data, dataPath(filenameJSON));
    emit conversionDone();
}
```

你可以加载文件或从您从网络请求得到的 ByteArray。 JsonDataAccess 或 XmlDataAccess 会给你的 QVariant。这的 QVariant 可以是 QVariantMap 如果内容是一个单一的对象，或 QVariantList 如果它是一个数组。

在示例应用程序,你会发现如何转换 XML 到 JSON 的样品,或从 JSON 到 XML。您还可以看到信号是如何发出的，以及如何将扬声器转换为一个更小的 JSON 文件与真快读访问。这里的 MeasureSpeakerPage：



原文链接：http://www.cnbbdevgroup.com/cn/?p=16504&utm_source=tuicool

常用 CSS 优化总结——网络性能与语法性能建议

在前端面试中最常见的问题就是页面优化和缓存（貌似也是页面优化），被问了几次后心虚的不行，平然平时多少会用到一些，但突然问我，很难把自己知道的都说出来。页面优化明显不是一两句能够说完的，这两天总结了一下 CSS 相关的优化知识，写篇博客梳理一下，还望大家多多指教。

关于 CSS 的优化工作主要从两个方面着手

- 网络性能：把 CSS 写到字节数最少，加快下载速度，自然可以让页面渲染的更快一些
- 语法性能：同样都能实现某些效果，但并不是所有的方式效果都相同，我们看过不少关于 JavaScript 方面的语法优化知识，其实 CSS 里面也有一些

CSS 压缩

CSS 压缩并不是什么高端的姿势，但却很有用，其原理很简单，就是把我们 CSS 中没用的空白符等删去，达到缩减字符个数的目的

我们有这样一段 CSS 脚本

```
    .test{    background-color:#ffffff;
background-image:url(a.jpg);
    }
```

经过压缩后会变成这样

```
.test{ background-color:#fff;  background-image:url(a.jpg)}
```

当然高级些的压缩工具也会帮我们优化一些语法，提供很多选项，让我们的压缩更有控制，之前在的公司不采用 CSS 压缩，所以我没有什么实践经验，自己写东西常用的是 YUI Compressor，有很多在线版的很方便

YUI Compressor

CSS Compressor

CSS drive

Clean CSS

大家有什么好的资源希望也推荐一下

gzip 压缩

Gzip 是一种流行的文件压缩算法，现在的应用十分广泛，尤其是在 Linux 平台，这个不止是对 CSS，当应用 Gzip 压缩到一个纯文本文件时，效果是非常明显的，大约可以减少70%以上的文件大小（这取决于文件中的内容）。想进一步了解 gzip 看看维基百科。

在没有 gzip 压缩的情况下，Web 服务器直接把 html 页面、CSS 脚本、js 脚本发送给浏览器，而支持 gzip 的 Web 服务器将把文件压缩后再发给浏览器，浏览器（支持 gzip）在本地进行解压和解码，并显示原文件。这样我们传输的文件字节数减少了，自然可以达到网络性能优化的目的。gzip 压缩需要服务器的支持，所以我们需要在服务器端进行配置

在 IIS 上启用 Gzip 压缩(HTTP 压缩)

apache 启用 gzip 压缩方法

Nginx Gzip 压缩配置

当然除了 gzip 压缩,缓存也是我们需要注意的,这和 CSS 优化关系不大了,在说 web 优化的时候再说

合写 CSS

除了压缩的方式,我们还可以通过少写 CSS 属性来达到减少 CSS 字节的目的,拿个最常见的例子

```
.test{ background-color: #000; background-image: url(image.jpg);
background-position: left top; background-repeat: no-repeat;
```

```
}
```

我们可以改写一下上面的 CSS，达到同样的效果

```
.test{  background: #000 url(image.jpg) top left no-repeat;
}
```

在 CSS 中还有很多类似的属性可以合写

font

```
{font-style: oblique; font-weight: bold; font-size: 16px;
font-family: Helvetica, Arial, Sans-Serif;}

{font: oblique bold 16px Helvetica, Arial, Sans-Serif;}
```

margin/padding

```
{margin-top: 5px; margin-right: 10px; margin-bottom: 20px;
margin-left: 15px;}

{margin: 5px 10px 20px 15px;}
{padding-top: 5px; padding-right: 10px; padding-bottom: 5px;
padding-left: 10px;}

{padding: 5px 10px}
{padding-top: 5px; padding-right: 5px; padding-bottom: 5px;
padding-left: 5px;}
{padding:5px;}
```

background

```
{background-color: #000; background-image: url(image.jpg);
background-position: left top; background-repeat: no-repeat;}
{background: #000 url(image.jpg) top left no-repeat;}
```

border

```
{border-width: 2px; border-style: solid; border-color: #000;}
```

```
    {border: 2px solid #000;}
    {border-top: 2px; border-right: 5px; border-left: 10px;
border-bottom: 3px;}
    {border: 2px 5px 10px 3px;}
```

另外 CSS3添加的很多属性如 transform、animation 相关的都可以合写，不一一列举，大家用的时候要注意

利用继承

CSS 的继承机制也可以帮我们再一定程度上缩减字节数，我们知道 CSS 有很多属性是可以继承的即在父容器设置了默写属性，子容器会默认也使用这些属性，因此如果我们希望全文字体尺寸是14px，大可不必为每个容器设置，只需要在 body 上设置就可以了。应用这个技巧，把 CSS 属性在可能的情况下提到父容器是可以帮我们节省 CSS 字节的，顺便说一下哪些属性可以继承

- 所有元素可继承：visibility 和 cursor
- 内联元素和块元素可继承：letter-spacing、word-spacing、white-space、line-height、color、font、 font-family、font-size、font-style、font-variant、font-weight、 text- decoration、 text-transform、 direction
- 块状元素可继承：text-indent 和 text-align
- 列表元素可继承：list-style、list-style-type、list-style-position、list-style-image
- 表格元素可继承：border-collapse
- 不可继承的：display、margin、border、padding、background、height、min-height、max- height、width、min-width、max-width、overflow、position、left、right、top、 bottom、z-index、float、clear、table-layout、vertical-align、page-break-after、 page-bread-before 和 unicode-bidi

css 中可以和不可以继承的属性

抽离、拆分 CSS，不加载所有 CSS

抽离 CSS 是指把一些通用的 CSS 放到一个文件内，而不是写道各个页面，这样一次下载后，其它页面用到的时候就可以利用缓存了，减少不必要的重复下载。

应用抽离原则，在很多时候我们把页面通用的 CSS 写到了一个文件，这样加

载一次后就可以利用缓存，但这样做并不适合所有场景，以前我写 CSS 把一些前端插件用的 CSS 全写到了一个页面，但是有时候页面只会用一个 Dialog、有的页面只用到了一个 TreeView，但却把十多个插件的 CSS 都下载到了页面，这就是问题了，所以虽然把 CSS 写道一个文件可以减少 http 请求，但像刚才的这种情况是不应该这样做的，对一些所有页面都会用到的我们可以这样做，所以我们在抽离和拆分的时候可要想好了。

CSS sprites

这个其实算不上是 CSS 优化，应该说是 web 优化用到了 CSS 的技巧，顺便提一下，有兴趣同学可以看看使用 CSS sprites 减少 HTTP 请求。

网络性能方面能想到的就暂时这么多了，希望大家帮忙指正和补充，看一些语法上的性能优化

CSS 放在 head 中，减少 repaint 和 reflow

相信做 web 的同学都知道这条建议，但为什么 CSS 放在页面顶部有利于网页优化呢？浏览器渲染页面大概是这样的，当浏览器从上到下一边下载 html 生成 DOM tree 一边根据浏览器默认及现有 CSS 生成 render tree 来渲染页面，当遇到新的 CSS 的时候下载并结合现有 CSS 重新生成 render tree，刚才的渲染工作就白费了，如果我们把所有 CSS 都放到页面顶部，这样就没有重新渲染的过程了。对浏览器工作原理有兴趣的同学可以看看神文

浏览器的工作原理：新式网络浏览器幕后揭秘，相信会对浏览器工作原理有深入的认识。

类似的我们知道了这个也应该在脚本中注意尽量减少 repaint 和 reflow，什么情况会导致这两种情况呢

reflow：当 DOM 元素出现隐藏 / 显示、尺寸变化、位置变化的时候都会让浏览器重新渲染页面，之前渲染工作白费了

repaint：当元素的背景颜色、边框颜色不引起 reflow 的变化是会让浏览器重新渲染该元素。貌似还可以接受，但如果我们在开始就定义好了，不让浏览器重复工作就更好了。

不用 CSS 表达式

无论怎样生成的 CSS，最终我们放到页面上得是静态普通文本，没有变量、

计算神马的，CSS 表达式是一种动态设置 CSS 属性的东东，被 IE5-IE8支持，看一个大家常用的例子

```
    body {  background-color: expression((new Date()).getHours()%2?"#B8D4FF":"#F08A00");
    }
```

这样我们赋予了 CSS 类似 JavaScript 的功能，CSS 表达式非常强大，甚至可以使用 CSS 表达式实现 min-width 属性，隔行换色，模拟 :hover, :before, :after 等伪类，看起来能解决很多 IE 下的浏览器兼容性等问题，但是其带来的副作用超出我们的想象，这条 CSS 规则并不是只运行一次，为了确保有效性，CSS 表达式会进行频繁的求值，当改变窗口大小，滚动页面甚至移动鼠标都会触发表达式进行求值，如此频繁的求值以至于浏览器的性能收到严重的影响。据《高性能网站建设建议》中的测试其执行次数远远超出我们想象，感兴趣同学可以进去看看，我们的建议就试尽量避免甚至不要使用 CSS 表达式。

不乱用 CSS reset 或属性设置

在网站建设中我们经常使用一些 CSS reset，达到跨浏览器统一的目的，但是很多时候我们的 CSS reset 过于臃肿，主要有两个问题

1 把很多浏览器对元素的默认属性有设置了一边，比如 div 的 padding 和 margin 为0啊什么的，这是没有必要的

2 把一些很不常用的元素的设置也写进了 CSS reset，如 ruby 这样的元素

避免适用通配符或隐式通配符

CSS 中的*代表通配符，虽然好用但使用不当这也是一个恶魔，比如

```
    body * {padding:0;margin:0;}
```

我们以为这是对 body 的子结点都设置一些属性，但因为 CSS 继承特性的原因，页面所有元素都会接受这个规则，对于复杂的页面在性能上的影响还是很大的，这并不是说不能使用通配符，而是说使用的时候要注意范围。相信这个规则大家都知道，但是有一些隐式的通配符也需要我们的注意，比如

```
    :visible{  padding:0;
    }
```

这样的几乎就和通配符一样，在使用的时候一定要注意范围限制问题

避免层级或过度限制的 CSS

估计 web 开发的同学都看过 MDN 上 Writing efficient CSS 或者其各种翻译版本，文中总结了几点在书写 CSS selector 的意见，搞明白文中建议的一个前提是得知道 CSS 是从右到左解析的，而不是我们认为的从左到右，关于为什么这样做肯定是因为高效，不明就里的同学可以上网搜一下相关知识。

抄袭一下文章内容

不要用标签或 class 来限制 ID 规则

这个应该是个常识，但很多同学都会误用，写出#test.info 或者 div#test 这样的选择器，这个只能说是画蛇添足，id 已经可以唯一而且最快的定位一个元素了

不要用标签名限制 class 规则

这个估计被误用的更多，如 div.info 这样的写法，其实我们可以直接写为.info，从右到左解析的原因可以知道为什么其低效，如果直接使用 class 不能达到目的，一般情况下应该是 class 设计的有问题，CSS 需要重构了

尽量使用最具体的类别、避免后代选择器、属于标签类别的规则*永远*不要包含子选择器

这三条规则是想通的，因为从左到右解析关系，在 CSS 选择器中后代选择器非但没有帮我们加快 CSS 查找，反而**后代选择器是 CSS 中耗费最昂贵的选择器。**它的耗费是极其昂贵的—**特别是当选择器在标签或通用类别中**，作者给的建议是**当使用子选择器时要十分谨慎，能免则免**。其开销可见一斑了。

对此我们可以通过具体类别——使用单一或尽量少的 class 解决。

最后

平常用到和了解的关于优化 CSS 达到页面优化的手段就这些了，希望大家给出意见和补充。

原文链接：http://www.cnblogs.com/dolphinX/p/3508657.html?utm_source=tuicool

[编程语言]

2013 流行 Python 项目汇总

Python 作为程序员的宠儿，越来越得到人们的关注，使用 Python 进行应用程序开发的越来也多。那么，在 2013 年有哪些流行的 Python 项目呢？下面，我们一起来看下。

一、测试和调试

- python_koans：Python Koans 算 “Ruby Koans” 的一部分，作为交互式教程，可以学习 TDD 技巧。
- sure：Sure 是最适合自动化测试的 Python 工具，包含流利的断言、深度选择器等等特性。
- responses：用 responses 能令测试更加轻松，这是一个可以伪装各种请求的库。
- boom：Boom! Apache Bench 的替代品。作为一个命令行工具，Boom 能对你的应用进行快捷的 smoke test。
- cricket：BeeWare 套件的一部分，cricket 是种图形化工具，协助你进行案例测试。
- bugjar：BeeWare 套件的一部分，bugjar 是针对 Python 的图形化交互式调试器。
- pudb：pudn 是针对 Python 的全屏命令行调试器。
- voltron：更好的 gdb 界面。

二、Web 框架

- django-stronghold：试过将 login_required 装饰器四处乱放？在你的堡垒中令所有 Django 视图有默认 login_required 呗。
- Falcon Framework：Falcon 自称为高性能云接口框架，号称能在相同硬件条件下提高服务端性能30倍！听起来有点儿意思？
- django-xadmin：用 bootstrap 对 django-admin 进行了深度升级，提供了可插件安装的仪表盘。
- clay：基于 Flask 的封装，能令我们轻松的创建 RESTful 后端服务，完

整文档可查看 clay。

- flask-restful：基于 Flask 的简单框架，用以创建 REST 接口。
- sandman：Sandman 希望通过 REST 接口暴露你现有的 app，相关 博客也值得一读。
- Django Unchained：名字很高大上，也的确是 Python Django 初学者的靠谱指南。

三、并发

- pulsar：部署新 web 服务器走起！有趣的事件驱动的并发框架 ！兼容从 2.6+到 pypy 的所有 python 版本！
- toro：同步化的 Tornado 协程支持。
- offset:Offset Go 的 并发模式在 Python 中的实现,请参考相关演讲 幻灯来理解!

四、任务调度

- pyres：从 resque 获得灵感的纯 Python 任务调度模块，是 celery 的替代。
- dagobah：Dagobah 是 Python 完成的简单关系依赖为基础的任务调度模块，还包含很 COOL 的关联任务工作流图形工具。
- schedule：使用生成器模式来为定期任务生成配置的进程调度模块。

五、实用工具

- howdoi：发觉你总在 Google 一些简单的最简单的编程任务？howdoi 能让你远离浏览器，就解决这类事儿！
- delorean：时间旅行?简单！Delorean 的目标就是令你的 Python 项目在处理时间/日期时轻而易举!查阅完备的 文档。
- powerline-shell：对于那些想让常用工具漂亮起来人，一定要用 powerline-bash，能打造漂亮的 shell 提示符，增加力线(powerline)，兼容 Bash/Zsh。
- fn.py：在 Python 中谈及函数编程时失落的那节"电池"终于出现了！如果对 Python 函数式编程有兴趣的立即下手安装体验吧！
- lice：为你的开源工程方便的追加许可证，而不用自个儿去 Google，支

持 BSD、MIT 和 GPL 以及变种协议。

- usblock：基于 USB 来锁定或是解锁你的笔记本！
- Matchbox：MatchBox 能在你自个儿的服务器上提供类似 Dropbox 风格的备份服务！基于 Flask 并通过 http 协议进行文件传输。
- cleanify：用 cleanify 能异步美化你项目的所有 html/css/js 文件。
- locksmith：Locksmith 是 AES 加密的口令管理器，看起来不错，完全开源，源代码、截屏都有。
- storm：在 Storm 的命令行界面，管理你所有的 SSH 连接。
- sqlparse:：这个很给力！sqlparse 是个 SQL 有效性分析器，支持解析/分裂/格式化 SQL 语句。
- autopep8：能自动化以 pep8来格式化你的代码。
- colout：colout 用以在命令行中色彩化输出，这就从其 github page 查看范例来体验吧。
- bumpversion：版本号冲撞总是恼人的，而每个人总是忘记给发行版本打 tag，bumpversion 用一条简单的命令简化了这方面的操作。
- pyenv：需要更好的管理你 Python 的多版本环境 ？ pyenv 让你能简洁的作到！(甚至超出你的预期！有插件能将 VirtualEnv 也无缝结合进来！)
- pip-tools：一整套能令你的 Python 项目保持清爽的工具。
- cdiff：Cdiff 是种非常 nice 的工具，可以用彩色输出统一 diff 格式信息，或用双栏形式来展示。

六、数据科学及可视化

- data_hacks：由 bitly 发布的一堆数据分析用命令行工具。这些工具接受命令行或是其它工具输入的数据，轻易的生成柱图以及直方图等等。
- 给黑客的概率编程和贝叶斯方法：这书是极好的，介绍如何用贝叶斯方法和概率编程进行数据分析，而且每章都提供了用以 iPython Notebooks 的示例。
- simmetrica：想对自个儿的应用基于时间的数据序列 进行展示、汇总、分享嘛？赶紧上 simmetrica 吧，同时还提供了可定制的仪表盘。
- vincent： Python 构建的专为运用 D3.js 进行可视化的 vega 转换工具。
- bamboo：一种简洁的实时数据分析应用，bamboo 提供了一个进行合并、

汇总、数值计算的数据实时接口。

- dataset：难以置信的工具，dataset 让对数据库的读写简单的象对 JSON 文件的操作，没有其它的文件配置，顷刻间就让你在 BOSS 面前高大上起来。
- folium：喜欢地图?也爱 Python? Folium 让你在地图上自在操纵数据。
- prettyplotlib：用 prettyplotlib 来强化你的 matplotlib，让你默认的 matplotlib 输出图片更加漂亮.
- lifelines：有兴趣在 Python 中研究生存分析的话，不用观望了，用 lifelines! 包含对 Kaplan-Meier、Nelson-Aalen 和生存回归分析。

七、编辑器及其改善

- sublime-snake：想在无尽的 coding 中喘口气? 当然是这种经典游戏了……
- spyderlib：又一个用 Python 写的开源 IDE。
- vimfox：对于 Vim 党最贴心的 web 专发工具，VimFox 能让 vim 实时的看到 css/js/html 的修改效果，能神奇的让 vim 中的修订，立即在浏览器中看到。
- pcode：基于 Py3 的 IDE，通过简单的 UI 提供了重构、工程管理等。

八、持续交付

- metrology：这个库很酷，支持你对应用进行多种测量并轻松输出给类似 graphite 的外部系统。
- python-lust：支持在 Unix 系统中用 Python 实现一个守护进程。
- scales：Scales 对你的 Python 应用进行持续状态和统计，并发送数据到 graphite。
- glances：跨平台，基于 curses 命令行的系统监视工具。
- ramona：企业级的应用监管。 Ramona 保证每个进程在值，一但需要立即重启，并有监控/日志输出，会发送邮件提醒。
- salmon：基于 Salt Stack 的多服务监视系统，即能作报警系统，也能当监控系统。
- graph-explorer：Graph-explorer 是对 Graphite 面板的增强，比原版的好很多，值得体验。
- sovereign：Sovereign 是一系列 ansible 的攻略手册，能为自个儿建造

个私人云。

- shipyard：能在指定的机器上弹出你的弹窗实例，也支持你创建/删除等等对弹窗的远程控制。
- docker-py：疯狂的 docker 工程接口的 Python 包装。
- dockerui：基于 docker 接口通过 web 界面进行交互操作的工具。
- django-docker：如果想知道怎么将 Djnago 应用同 Docker 结合？可以从这里学习。
- diamond：Python 实现的守护进程，自动从你的服务或是其它指定数据源中提取数值，并 graphite、以及其它支持的状态面板/收集系统输出。

九、Git

- git-workflow：可视化你的 git 工作流程的工具，示例： Demo。
- gitto：简洁的库，协助你建立自个儿的 git 主机。
- git-imerge：git-imerge 能让 git 进行增量合并。 本质上是允许你在进行 imerge 有冲突时，有机会先合并掉，再继续。

十、邮件与聊天

- mailbox：Mailbox 是对 Python 的 IMAP 一个人性化的再造。 基于简单即是美的态度，作者对 IMAP 接口给出了一个简单又好理解的形式。
- deadchat：deadchat 旨在不安全的网络环境中提供安全的单一房间群聊服务以及客户端。
- Mailpile：Mailpile 是针对邮件的索引及搜索引擎 。

十一、音频和视频

- pms：穷人的 Spotify，搜索和收集音乐流！
- dejavu：在琢磨 Shazam 的原理？音频指纹识别算法的 Python 实现在此！（译注:Shazam：是个神奇的音乐识别应用，对她哼个几秒调子，就能精确告诉你是什么歌曲、作者、歌词……）
- HTPC-Manager：为 HTPC 粉丝准备的工具，提供了完备的界面来管理所有家庭媒体服务器上的好物。
- cherrymusic：Python 实现的一个音乐流媒体服务器。 流化输出你的音乐到所有设备上。

- moviepy：脚本化的电影剪辑包，切/串/插入标题等基本操作，几行就搞定！

十二、其它

- emit：用 redis 为你的函式追加可订阅能力，很有趣。
- zipline：Zipline 是种 很 Pythonic 的交易算法库。
- raspberry.io：Raspberry.io 是树莓派的社区实现。 刚刚发布，汇集了各种创意想法，有兴趣的话立即检出折腾吧。
- NewsBlur：Google Reader 已经关张儿了，Newsblur 已经发布了有段日子了，开源的 RSS 阅读器，这绝对是应该首先体验的。
- macropy：Macropy 是在 Python 中实现 macros 的库。 检出文档，参考所有功能，看怎么用上了。
- mini：对编译器以及语言设计有兴趣的，一定要看看这个仓库，以及配套的录像！
- parsimonious：Parsimonious 的目标 是最快的 arbitrary-lookahead 解析器。 用 Python 实现，基本可用。
- isso：Disqus 的开源替代，从 demo 看很不错，而且提供了更好的隐私设置。
- deaddrop：Deaddrop 能为新闻机构或是其它人 提供在线投递箱，详细信息参考其 github page。
- nude.py：裸体检测的 Python 实现，是 node.js 的仿制。
- kaptan： Kaptan 是你应用的配置管理器！
- luigi：Luigi 帮你构建复杂的管道来完成批处理。
- gramme：Gramme 以简单而优雅的方式，通过 UDP 接口对易失数据完成消息包装序列化。
- q：为你的 Python 程序提供快速而随性的日志。 有一系列帮手来追踪你的函式参数，并能在控制台快速交互式加载。
- fuqit：来自伟大的 Zed Shaw 最新作品，fuqit 试图令你忘记 MVC 的经验，用全新的方式专注简洁一切。
- simplicity：基于靠谱的 pydanny 将你的 新结构化文本 转换为 JSON

格式。

- lassie：Lassie 允许你轻松的从网站检索出内容来。
- paperwork：Paperwork 是个 OCR 文档并完成可搜索转化的工具，用 GTK/Glade 实现了友好的界面。
- cheat：cheat 允许你创建并查阅命令行上的交互式备忘。设计目的是帮助 *nix 的系统管理员们在习惯的环境中，快速调阅不易记忆的常用命令。
- cookiecutter：良心模块！提供一堆有用但是不常写，所以记不下来的代码模板，也支持自制代码模板。
- pydown：支持用 Python 构建亮丽的 HTML5 效果幻灯，Demo。
- Ice：模拟器粉丝们现在能用 Ice 向 Steam 里塞 ROM 来玩了。
- pants：用以编写异步网络应用的轻量级框架。 Pants 是单线程，回调服务，也包含支持 Websockets 的 HTTP 服务、WSGI 支持和一个简单的 web 框架。
- pipeless：Pipeless 是一个构建简单 数据管道的框架。
- marshmallow：marshmallow 是个 ORM 无关的库，能将复杂的数据类型转换为 Python 原生类型对象，以便容易的转换为 JSON 提供接口使用。
- twosheds：Python 实现的库，用来构造命令或是 shell 解释器。Twosheds 让你用 Python 来定制自个儿的 shell 环境。

原文链接 http://www.iteye.com/news/28717-2013-top-python-projects?utm_source=tuicool

15款 Django 开发常用软件包

Django 是一款高级的 Python Web 框架，可以帮助开发者快速创建 web 应用。我们这里整理了 15 款 Django 开发中常用的软件包，学会使用它们可以节省大量开发时间，提高开发效率。同时，也给出了它们的 pip 安装方法。下面一起来看下。

一、认证和授权

1. Python social auth

一款社交账号认证/注册机制，支持 Django、Flask、Webpy 等在内的多个开发框架，提供了约 50 多个服务商的授权认证支持，如 Google、Twitter、新浪微博等站点，配置简单。

Shell 代码

```
• pip install python-social-auth
```

2. Django Guardian

Django 默认没有提供对象（Object）级别的权限控制，我们可以通过该扩展来帮助 Django 实现对象级别的权限控制。

Shell 代码

```
• pip install django-guardian
```

3. Django OAuth Toolkit

可以帮助 Django 项目实现数据、逻辑的 OAuth2 功能，可与 Django REST 框架完美整合起来。

Shell 代码

```
• pip install django-oauth-toolkit
```

4. django-allauth

可用于账号注册、管理和第三方社交账号的认证。

Shell 代码

```
• pip install django-allauth
```

二、后端

1. Celery

用来管理异步、分布式的消息作业队列，可用于生产系统来处理百万级别的任务。

Shell 代码

```
• pip install Celery
```

2. Django REST 框架

构建 REST API 的优秀框架，可管理内容协商、序列化、分页等，开发者可以在浏览器中浏览构建的 API。

Shell 代码

```
• pip install djangorestframework
```

3. Django stored messages

可以很好地集成在 Django 的消息框架中（django.contrib.messages）并让用户决定会话过程中存储在数据库中的消息。

4. django-cors-headers

一款设置 CORS（Cross-Origin Resource Sharing）标头的应用，基于 XmlHttpRequest，对管理 Django 应用中的跨域请求非常有帮助。

Shell 代码

```
• pip install django-cors-headers
```

三、调试

1. Debug toolbar

可在设置面板显示当前请求/响应的各种调试信息。除了本身提供的操作面板外，还有来自社区的多个第三方面板。

Shell 代码

```
• pip install django-debug-toolbar
```

四、静态资源

1. Django Storages

可使静态资源方便地存储在外部服务上。安装后只需运行“python manage.py collectstatic”命令就可以将全部改动的静态文件复制到选定的后端。可结合库“python-boto”一起使用，将静态文件存储到 Amazon S3 上。

Shell 代码

```
• pip install django-storages
```

2. Django Pipeline

静态资源管理应用，支持连接和压缩 CSS/Javascript 文件、支持 CSS 和 Javascript 的多种编译器、内嵌 JavaScript 模板，可充分允许自定义。

Shell 代码

```
• pip install django-pipeline
```

3. Django Compressor

可将页面中链接的以及直接编写的 JavaScript 和 CSS 打包到一个单一的缓存文件中，以减少页面对服务器的请求数，加快页面的加载速度。

Shell 代码

```
• pip install django_compressor
```

五、工具

1. Reversion

为模型提供版本控制功能，稍微配置后，就可以恢复已经删除的模型或回滚到模型历史中的任何一点。最新版本支持 Django 1.6。

Shell 代码

```
• pip install django-reversion
```

2. Django extensions

Django 框架的扩展功能集合，包括 management 命令扩展、数据库字段扩展、admin 后台扩展等。

Shell 代码

```
• pip install django-extensions
```

3. Django braces

是一系列可复用的行为、视图模型、表格和其他组件的合集。

Shell 代码

```
• pip install django-braces
```

原文链接： http://www.iteye.com/news/28697?utm_source=tuicool

Rails 3 升级 Rails 4 中遇到的问题及解决方法

有些出现的问题其实是不懂正确的流程，都是在试错，可是还是学到了很多东西，写下了，希望对我和大家都有帮助。

Homebrew 的问题

当我去运行 `brew update` 的时候出现错误 `untracked working tree files`，因为 homebrew 是用 `Git` 去更新的，所以如果目录中出现 untracked files 就会导致不能更新。然后我看了 homebrew 的 `Common Issues` 文档。

解决方法

其实我对 Git 还算了解，可是就不知道 homebrew 的 working tree files 在哪里，所以下面的东西就直接解决了我的问题。

This is caused by an old bug in the update code that has long since been fixed. However, the nature of the bug requires that you do the following:

```
cd $(brew --repository)
git reset --hard FETCH_HEAD
```

If brew doctor still complains about uncommitted modifications, also run this command:

```
cd $(brew --repository)
git clean -fd
```

PostgreSQL 的问题

当出现 pg gem 不能 `bundle install` 的时候，我也尝试过 `gem install pg --with-pg-config` 这种提示里面的命令，可是还是不能解决这个问题。然后我就用 homebrew 把 postgresql 从9.2.3升级到了9.3.2

后果

这样做的直接后果就是 postgresql 不能正常启动，出现了一下的提示信息：

```
FATAL: database files are incompatible with server DETAIL: The da
```

```
ta directory was initialized by PostgreSQL version 9.2, which is not
compatible with this version 9.3.2.
```

原来 postgresql 升级以后不能兼容原来的数据文件，就是个悲剧啊。看了一下 postgresql 的升级文档，PostgreSQL major versions are represented by the first two digit groups of the version number，原来前两位数字都是主版本号。

解决方法

一般自己机器上面的都是测试数据，所以可以直接删除掉旧的数据库文件。运行一下命令就可以了。

```
rm -rf /usr/local/var/postgres
initdb -D /usr/local/var/postgres
```

如果你想要以前的数据文件，特别如果遇到在 production server 上升级了 postgresql，那么你就需要使用 `pg_dump` 出原来的数据文件，然后就要用到 `pg_upgrade` 啦。具体方式可以查看 `pg_upgrade` 的文档。

Rails Gem PG 的问题

这个时候 pg 已经成功安装成功了，可是在 `rake db:create` 的时候又出现关于 postgresql 的问题了：

```
Library not loaded: libpq.5.6.dylib
```

凭借自己的经验，觉得应该是 postgresql 中 lib 的这一个文件没有被 rake 的时候加载到。

解决方法

```
ln -s /usr/local/Cellar/postgresql/9.3.2/lib/libpq.5.6.dylib /usr
/local/lib/libpq.5.6.dylib
```

然后就可以该干嘛干嘛了。

原文链接：http://blog.segmentfault.com/ericwu/1190000000388741?utm_source=tuicool

php 性能优化

很长时间没有写博文了，最近换了工作，长时间加班，根本没有时间做其他事情！今天闲下来了，想一想 php 性能方面的事情。这也是我2014年的第一篇博文！

php 是一个很流行的脚本语言，现在很多公司（新浪、优酷、百度、搜狐、淘宝等等）在使用这种语言进行网站开发。我的这篇文章，我只是希望能够提高你的 php 脚本性能。请记住你的 php 脚本性能，很多时候依赖于你的 php 版本、你的 web server 环境和你的代码的复杂度。

优化你代码中的瓶颈

Hoare 曾经说过“过早优化是一切不幸的根源”。当你想要让你的网站更快运转的时候，你才应该去做优化的事情。当你要改变你代码之前，你需要做的事是什么原因引起了系统缓慢？你可以通过以下指导和其他方式优化你的 php，可能是数据库原因也可能是网路原因!通过优化你的 php 代码，你能尝试着找出你的系统瓶颈。

升级你的 php 版本

你的团队成员提出，这些年 php 引擎已经有很多象征性的性能提升。如果你的 web server 仍然运行着比较老的版本，如 php3或者 php4。那么在你尝试着优化你代码之前，应该先深入调查一下版本之间的升级情况。

点击以下链接，可以了解具体细节:

从 PHP 4 移植到 PHP 5

从 PHP 5.0.x 移植到 PHP 5.1.x

从 PHP 5.1.x 移植到 PHP 5.2.x

使用缓存

利用缓存模块（如 Memcache）或者模板系统(如 Smarty)进行缓存处理。我们可以缓存数据库结果和提取页面结果的方式来提升网站性能。

使用输出缓冲区

当你的脚本尝试着渲染的时候，php 会使用内存缓存区保存所有的数据。缓存区可能让你的页面看起来很慢，原因是缓冲区填满所有要响应的数据之后再把

结果响应给用户。幸运的是，你能够做一下改变，迫使 php 强行在缓冲区填满之前把数据响应给用户，这样就会让你的网站看起来更快一些。

- 输出缓存控制

避免写幼稚的 setters 和 getters

当你写 php 类的时候，你可以直接操作对象属性，这样能帮助你节省时间和提升你的脚本性能。而不是那种让人感到幼稚可笑的 setters 和 getters。

下面是一些案例：dog 类通过使用 setName() 和 getName() 方式来操作 name 属性。

```
1  class dog {
2    public $name ='';
3
4    public function setName($name) {
5      $this->name =$name;
6    }
7
8    public function getName() {
9      return $this->name;
10   }
11 }
```

注意：setName() 和 getName() 除了存储和返回 name 属性外，没做任何工作。

```
1  $rover =new dog();
2  $rover->setName('rover');
3  echo $rover->getName();
```

```
1  $rover =new dog();
2  $rover->name ='rover';
3  echo $rover->name;
```

直接设置和访问 name 属性，性能能提升100%，而且也能缩减开发时间！

没有原因不要 copy 变量

有时初级 phper，为了使代码更加“干净”，常常把已经定义的变量重新赋值给另一个变量。这实际上就导致了双重内存的消耗（当改变变量的时候），这就导致脚本的性能下降。比如一个用户把一个512KB 的变量在额外插入给另一个变量，那么就会导致1MB 的内存被消耗掉。

```
1
2
```

```
$description
=strip_tags($_POST['description']);
echo $description;
```

上面的代码没有任何原因，复制了一遍变量。你仅需要使用内联的方式简单输出变量，而不用额外的消耗内存。

```
1
```

```
echo strip_tags($_POST['description']);
```

避免循环做 SQL 操作

经常犯的错误是把一个 SQL 操作放置到一个循环中，这就导致频繁的访问数据库，更重要的是，这会直接导致脚本的性能低下。以下的例子，你能够把一个循环操作重置为一个单一的 SQL 语句。

```
1
2
3
4
```

```
foreach ($userList as $user) {
$query ='INSERT INTO users (first_name,last_name)
VALUES("' .$user['first_name'] .'",
"' .$user['last_name'] .'")';
mysql_query($query);
}
```

```
过程：
1
```

```
INSERT INTO users (first_name,last_name)
VALUES("John","Doe")
```

原文：http://www.cnblogs.com/baochuan/p/3523677.html?utm_source=tuicool

Java 中的 equals() 和 hashCode() 契约

java.lang.Object 类中有两个非常重要的方法：

```
public boolean equals(Object obj)
public int hashCode()
```

理解这两个方法非常的重要，尤其是将用户自定义的对象添加到 Map 中的时候。有时候就算是久经沙场的老程序员也弄不清楚该如何正确使用它们。这篇文章中，我将用一个例子让大家看看大家经常会犯的错误，然后解释 equals()和 hashCode()的正确的使用方法。

1. 常见错误

常见的错误如下：

```
import java.util.HashMap;
public class Apple {
    private String color;
    public Apple(String color) {
        this.color = color;
    }
    public boolean equals(Object obj) {
        if (!(obj instanceof Apple))
            return false;
        if (obj == this)
            return true;
        return this.color == ((Apple) obj).color;
    }
    public static void main(String[] args) {
        Apple a1 = new Apple("green");
        Apple a2 = new Apple("red");
        //hashMap stores apple type and its quantity
        HashMap m = new HashMap();
```

```
            m.put(a1, 10);
            m.put(a2, 20);
            System.out.println(m.get(new Apple("green")));
        }
    }
```

这个例子中，一个“绿苹果”的对象成功添加到 hashMap 中了，但是当我们要取出这个“绿苹果”的时候，却得不到这个对象，程序返回 null。我们使用调试器却发现在 hashMap 中已经存储了这个对象。

2. hashCode()引起的问题

这个问题是因为”hashCode()”方法没有被重写。Java 中 equals()和 hashCode()有一个契约：

- 如果两个对象相等的话，它们的 hash code 必须相等；
- 但如果两个对象的 hash code 相等的话，这两个对象不一定相等。

Map 的结构能够快速找到一个对象，而不是进行较慢的线性查找。使用 hash 过的键来定位对象分两步。Map 可以看作是数组的数组。第一个数组的索引就是对键采用 hashCode()计算出来的值，再在这个位置上查找第二个数组，使用键的 equals()方法来进行线性查找，直到找到要找的对象。

Object 类中的 hashCode()对于不同的对象返回不同的整数，所以上面的例子中，不同的对象（即使相同的类型）也返回不同的 hash 值。

Hash 码就像是一个存储空间的序列，不同的东西放在不同的存储空间中。将不同的东西整理放在不同的空间中（而不是堆积在一个空间中）更高效。所以能够均匀的分散 hash 码是再好不过了。

上面错误的解决方法就是在类中增加 hashCode 方法。这里我仅仅使用颜色的长度来计算 hash 码。

```
    public int hashCode(){
        return this.color.length();
    }
```

原文链接： http://segmentfault.com/a/1190000000389743?utm_source=tuicool

[程序设计]

IOS 缓存机制详解

为什么要有缓存

应用需要离线工作的主要原因就是改善应用所表现出的性能。将应用内容缓存起来就可以支持离线。我们可以用两种不同的缓存来使应用离线工作。第一种是**按需缓存**，这种情况下应用缓存起请求应答，就和 Web 浏览器的工作原理一样；第二种是**预缓存**，这种情况是缓存全部内容（或者最近 n 条记录）以便离线访问。

像第14章中开发的 Web 服务应用利用按需缓存技术来改善可感知的性能而不是提供离线访问。离线访问只是无心插柳的结果。Twitter 和 Foursquare 就是很好的例子。这类应用得到的数据通常很快就会过时。对于一条几天前的推文或者朋友上周在哪里你能有多大兴趣？一般来说，一条推文或者一条签到的信息只在几个小时内有意义，而24小时之后就变得无关紧要。不过大部分 Twitter 客户端还是会缓存推文，而 Foursquare 的官方客户端在无网络连接的情况下打开，会显示上次的状态。

大家可以用自己喜欢的 Twitter 客户端来试一下，Twitter for iPhone、Tweetbot 或其他应用：打开某个朋友的个人资料并浏览他的时间线。应用会获取时间线并填充页面。加载时间线时会看到一个表示正在加载的圆圈在旋转。现在进入另一个页面，然后再回来打开时间线。你会发现这次是瞬间加载的。应用还是在后台刷新内容（在上次打开的基础上），但是它会显示上次缓存的内容而不是无趣地转圈，这样看起来就快多了。如果没有缓存，用户每次打开一个页面都会看到圆圈在旋转。无论网络连接快还是慢，减小网络加载慢的影响，让它看起来很快，是 iOS 开发者的责任。这就能大大改善用户满意度，从而提高了应用在 App Store 中的评分。

另一种缓存更加重视被缓存数据，并且能快速编辑被缓存的记录而无需连接到服务器。代表应用包括 Google Reader 客户端，稍后阅读类的应用 Instapaper 等。

缓存的策略：

上一节中讨论到按需缓存和预缓存，它们在设计和实现上有很大的不同。按需缓存是指把从服务器获取的内容以某种格式存放在本地文件系统，之后对于每次请求，检查缓存中是否存在这块数据，只有当数据不存在（或者过期）的情况下才从服务器获取。这样的话，缓存层就和处理器的高速缓存差不多。获取数据的速度比数据本身重要。而预缓存是把内容放在本地以备将来访问。对预缓存来说，数据丢失或者缓存不命中是不可接受的，比方用户下载了文章准备在地铁上看，但却发现设备上不存在这些文章。

像 Twitter、Facebook 和 Foursquare 这样的应用属于按需缓存，而 Instapaper 和 Google Reader 等客户端则属于预缓存。

实现预缓存可能需要一个后台线程访问数据并以有意义的格式保存，以便本地缓存无需重新连接服务器即可被编辑。编辑可能是“标记记录为已读”或“加入收藏”，或其他类似的操作。这里**有意义的格式**是指可以用这种方式保存内容，不用和服务器通信就可以在本地作出上面提到的修改，并且一旦再次连上网就可以把变更发送回服务器。这种能力和 Foursquare 等应用不同，虽然使用后者你能在无网络连接的情况下看到自己是哪些地点的地主（Mayor），当然前提是进行了缓存，但无法成为某个地点的地主。Core Data（或者任何结构化存储）是实现这种缓存的一种方式。

按需缓存工作原理类似于浏览器缓存。它允许我们查看以前查看或者访问过的内容。按需缓存可以通过在打开一个视图控制器时按需地缓存数据模型（创建一个数据模型缓存）来实现，而不是在一个后台线程上做这件事。也可以在一个 URL 请求返回成功（200 OK）应答时实现按需缓存（创建一个 URL 缓存）。两种方法各有利弊，稍后我会在24.3节和24.6节中解释各个方法的优缺点。

选择使用按需缓存还是预缓存的一个简便方法是判断是否需要在下载数据之后处理数据。后期处理数据可能是以用户产生编辑的形式，也可能是更新下载的数据，比如重写 HTML 页面里的图片链接以指向本地缓存图片。如果一个应用需要做上面提到的任何后期处理，就必须实现预缓存。

存储缓存：

第三方应用只能把信息保存在应用程序的沙盒中。因为缓存数据不是用户产

生的，所以它应该被保存在 NSCachesDirectory，而不是 NSDocumentsDirectory。为缓存数据创建独立目录是一项不错的实践。在下面的例子中，我们将在 Library/caches 文件夹下创建名为 MyAppCache 的目录。可以这样创建：

```
NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSCachesDirectory,
        NSUserDomainMask, YES);
    NSString *cachesDirectory = [paths objectAtIndex:0];
    cachesDirectory = [cachesDirectory
        stringByAppendingPathComponent:@"MyAppCache"];
```

把缓存存储在缓存文件夹下的原因是 iCloud（和 iTunes）的备份不包括此目录。如果在 Documents 目录下创建了大尺寸的缓存文件，它们会在备份的时候被上传到 iCloud 并且很快就用完有限的空间（写作本书时大约为5 GB）。你不会这么干的——谁不想成为用户 iPhone 上的良民？NSCachesDirectory 正是解决这个问题的。

预缓存是用高级数据库（比如原始的 SQLite）或者对象序列化框架（比如 Core Data）实现的。我们需要根据需求认真选择不同的技术。本节第5点“应该用哪种缓存技术”给出了一些建议：什么时候该用 URL 缓存或者数据模型缓存，而什么时候又该用 Core Data。接下来先看一下数据模型缓存的实现细节。

1. 实现数据模型缓存

可以用 NSKeyedArchiver 类来实现数据模型缓存。为了把模型对象用 NSKeyedArchiver 归档，模型类需要遵循 NSCoding 协议。

NSCoding 协议方法

```
- (void)encodeWithCoder:(NSCoder *)aCoder;
- (id)initWithCoder:(NSCoder *)aDecoder;
```

当模型遵循 NSCoding 协议时，归档对象就很简单，只要调用下列方法中的一个：

```
[NSKeyedArchiver archiveRootObject:objectForArchiving
    toFile:archiveFilePath];
```

```
        [NSKeyedArchiver
archivedDataWithRootObject:objectForArchiving];
```

第一个方法在 archiveFilePath 指定的路径下创建一个归档文件。第二个方法则返回一个 NSData 对象。NSData 通常更快，因为没有文件访问开销，但对象保存在应用的内存中，如果不定期检查的话会很快用完内存。在 iPhone 上定期缓存到闪存的功能也是不明智的，因为跟硬盘不同，闪存读写寿命是有限的。开发者得尽可能平衡好两者的关系。24.3节会详细介绍归档实现缓存。

NSKeyedUnarchiver 类用于从文件（或者 NSData 指针）反归档模型。根据反归档的位置，选择使用下面两个类方法。

```
[NSKeyedUnarchiver unarchiveObjectWithData:data];
[NSKeyedUnarchiver unarchiveObjectWithFile:archiveFilePath];
```

这四个方法在转化序列化数据时能派上用场。

使用任何 NSKeyedArchiver/NSKeyedUnarchiver 的前提是模型实现了 NSCoding 协议。不过要做到这一点很容易，可以用 Accessorizer 类工具自动实现 NSCoding 协议。（24.8节列出了 Accessorizer 在 Mac App Store 中的链接。）

下一节会解释预缓存策略。我们刚才已经了解到预缓存需要用到更结构化的数据格式，接下来看看 Core Data 和 SQLite。

2. Core Data

正如 Marcus Zarra 所说，Core Data 更像是一个对象序列化框架，而不仅仅是一个数据库 API：

大家误认为 Core

Data 是一个 Cocoa 的数据库 API……其实它是个可以持久化到磁盘的对象框架（Zarra，2009年）。

要深入理解 Core Data，看一下 Marcus S. Zarra 写的*Core Data: Apple's API for Persisting Data on Mac OS X*（Pragmatic Bookshelf, 2009. ISBN 9781934356326）。

要在 Core Data 中保存数据，首先创建一个 Core Data 模型文件，并创建实

体（Entity）和关系（Relationship）；然后写好保存和获取数据的方法。应用可以借助 Core Data 获取真正的离线访问功能，就像苹果内置的 Mail 和 Calendar 应用一样。实现预缓存时必须定期删除不再需要的（过时的）数据，否则缓存会不断增长并影响应用的性能。同步本地变更是通过追踪变更集并发送回服务器实现的。变更集的追踪有很多算法，我推荐的是 Git 版本控制系统所用的（此处没有涉及如何与远程服务器同步缓存，这不在本书讨论范围之内）。

3. 用 Core Data 实现按需缓存

尽管从技术上讲可以用 Core Data 来实现按需缓存，但我不建议这么做。Core Data 的优势是不用反归档完整的数据就可以独立访问模型的属性。然而，在应用中实现 Core Data 带来的复杂度抵消了优势。此外，对于按需缓存实现来说，我们可能并不需要独立访问模型的属性。

4. 原始的 SQLite

可以通过链接 libsqlite3的库来把 SQLite 嵌入应用，但是这么做有很大的缺陷。所有的 sqlite3库和对象关系映射（Object Relational Mapping，ORM）机制几乎总是会比 Core Data 慢。此外，尽管 sqlite3本身是线程安全的，但是 iOS 上的二进制包则不是。所以除非用定制编译的 sqlite3库（用线程安全的编译参数编译），否则开发者就有责任确保从 sqlite3读取数据或者往 sqlite3写入数据是线程安全的。Core Data 有这么多特性而且内置线程安全，所以我建议在 iOS 中尽量避免使用 SQLite。

唯一应该在 iOS 应用中用原始的 SQLite 而不用 Core Data 的例外情况是，资源包中有应用程序相关的数据需要在所有应用支持的第三方平台上共享，比如说运行在 iPhone、Android、BlackBerry 和 Windows Phone 上的某个应用的位置数据库。不过这也不是缓存了。

---

5. 应该用哪种缓存技术

在众多可以本地保存数据的技术中，有三种脱颖而出：URL 缓存、数据模型缓存（利用 NSKeyedArchiver）和 Core Data。

假设你正在开发一个应用，需要缓存数据以改善应用表现出的性能，你应该

实现按需缓存（使用数据模型缓存或 URL 缓存）。另一方面，如果需要数据能够离线访问,而且具有合理的存储方式以便离线编辑,那么就用高级序列化技术(如 Core Data)。

6. 数据模型缓存与 URL 缓存

按需缓存可以用数据模型缓存或 URL 缓存来实现。两种方式各有优缺点，要使用哪一种取决于服务器的实现。URL 缓存的实现原理和浏览器缓存或代理服务器缓存类似。当服务器设计得体，遵循 HTTP 1.1的缓存规范时，这种缓存效果最好。如果服务器是 SOAP 服务器（或者实现类似于 RPC 服务器或 RESTful 服务器），就需要用数据模型缓存。如果服务器遵循 HTTP 1.1缓存规范，就用 URL 缓存。数据模型缓存允许客户端（iOS 应用）掌控缓存失效的情形，当开发者实现 URL 缓存时，服务器通过 HTTP 1.1的缓存控制头控制缓存失效。尽管有些程序员觉得这种方式违反直觉，而且实现起来也很复杂（尤其是在服务器端），但这可能是实现缓存的好办法。事实上，MKNetworkKit 提供了对 HTTP 1.1缓存标准的原生支持。

数据模型缓存：

本节我们来给第14章中的 iHotelApp 添加用数据模型缓存实现的按需缓存。按需缓存是在视图从视图层次结构中消失时做的（从技术上讲，是在 viewWillDisappear:方法中）。支持缓存的视图控制器的基本结构如图24-1所示。AppCache Architecture 的完整代码可从本章的下载源代码中找到。后面讲解的内容假设你已经下载了代码并且可以随时使用。

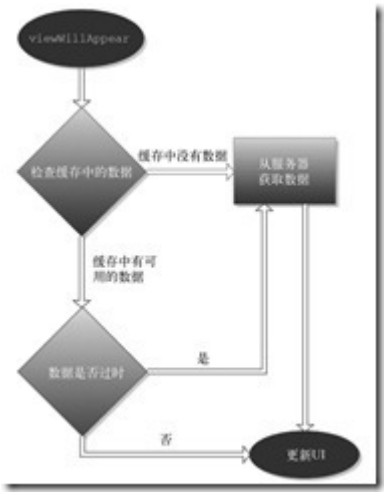


图24-1

实现了按需缓存的视图控制器的控制流

在 viewWillAppear 方法中，查看缓存中是否有显示这个视图所需的数据。如果有就获取数据，再用缓存数据更新用户界面。然后检查缓存中的数据是否已经过期。你的业务规则应该能够确定什么是新数据、什么是旧数据。如果内容是旧的，把数据显示在 UI 上，同时在后台从服务器获取数据并再次更新 UI。如果缓存中没有数据，显示一个转动的圆圈表示正在加载，同时从服务器获取数据。得到数据后，更新 UI。

前面的流程图假定显示在 UI 上的数据是可以归档的模型。在 iHotelApp 的 MenuItem 模型中实现 NSCoding 协议。NSKeyedArchiver 需要模型实现这个协议，如下面的代码片段所示。

MenuItem 类的 encodeWithCoder 方法（MenuItem.m）

```
- (void)encodeWithCoder:(NSCoder *)encoder
    {
        [encoder encodeObject:self.itemId forKey:@"ItemId"];
        [encoder encodeObject:self.image forKey:@"Image"];
        [encoder encodeObject:self.name forKey:@"Name"];
        [encoder encodeObject:self.spicyLevel
forKey:@"SpicyLevel"];
        [encoder encodeObject:self.rating forKey:@"Rating"];
        [encoder encodeObject:self.itemDescription
forKey:@"ItemDescription"];
        [encoder encodeObject:self.waitingTime
forKey:@"WaitingTime"];
        [encoder encodeObject:self.reviewCount
forKey:@"ReviewCount"];
    }
```

MenuItem 类的 initWithCoder 方法（MenuItem.m）

```
- (id)initWithCoder:(NSCoder *)decoder
    {if ((self = [super init])) {
            self.itemId = [decoder decodeObjectForKey:@"ItemId"];
            self.image = [decoder decodeObjectForKey:@"Image"];
            self.name = [decoder decodeObjectForKey:@"Name"];
            self.spicyLevel = [decoder
decodeObjectForKey:@"SpicyLevel"];
            self.rating = [decoder decodeObjectForKey:@"Rating"];
            self.itemDescription = [decoder
              decodeObjectForKey:@"ItemDescription"];
            self.waitingTime = [decoder
decodeObjectForKey:@"WaitingTime"];
            self.reviewCount = [decoder
decodeObjectForKey:@"ReviewCount"];
        }
    return self;
    }
```

就像之前提到过的，可以用 Accessorizer 来生成 NSCoding 协议的实现。

根据图24-1中的缓存流程图，我们需要在 viewWillAppear:中实现实际的缓存逻辑。把下面的代码加入 viewWillAppear:就可以实现。

视图控制器的 viewWillAppear:方法中从缓存恢复数据模型对象的代码片段

```
    NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSCachesDirectory,
        NSUserDomainMask, YES);
    NSString *cachesDirectory = [paths objectAtIndex:0];
    NSString *archivePath = [cachesDirectory
stringByAppendingPathComponent:@"AppCache/MenuItems.archive"];
```

```
        NSMutableArray *cachedItems = [NSKeyedUnarchiver
            unarchiveObjectWithFile:archivePath];
    if(cachedItems == nil)
          self.menuItems = [AppDelegate.engine localMenuItems];
    else
          self.menuItems = cachedItems;
        NSTimeInterval stalenessLevel = [[[[NSFileManager
defaultManager]
            attributesOfItemAtPath:archivePath error:nil]
        fileModificationDate] timeIntervalSinceNow];
    if(stalenessLevel > THRESHOLD)
          self.menuItems = [AppDelegate.engine localMenuItems];
        [self updateUI];
```

缓存机制的逻辑流如下所示。

- 视图控制器在归档文件 MenuItems.archive 中检查之前缓存的项并反归档。
- 如果 MenuItems.archive 不存在，视图控制器调用方法从服务器获取数据。
- 如果 MenuItems.archive 存在，视图控制器检查归档文件的修改时间以确认缓存数据有多旧。如果数据过期了（由业务需求决定），再从服务器获取一次数据。否则显示缓存的数据。

接下来，把下面的代码加入 viewDidDisappear 方法可以把模型（以 NSKeyedArchiver 的形式）保存在 Library/Caches 目录中。

视图控制器的 viewWillDisappear:方法中缓存数据模型的代码片段

```
    NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSCachesDirectory,
          NSUserDomainMask, YES);
        NSString *cachesDirectory = [paths objectAtIndex:0];
```

```
        NSString *archivePath = [cachesDirectory
stringByAppendingPathComponent:@"    AppCache/MenuItems.archive"];
        [NSKeyedArchiver archiveRootObject:self.menuItems
toFile:archivePath];
```

视图消失时要把 menuItems 数组的内容保存在归档文件中。注意，如果不是在 viewWillAppear:方法中从服务器获取数据的话，这种情况不能缓存。

所以，只需在视图控制器中加入不到10行的代码（并将 Accessorizer 生成的几行代码加入模型），就可以为应用添加缓存支持了。

重构

当开发者有多个视图控制器时，前面的代码可能会有冗余。我们可以通过抽象出公共代码并移入名为 AppCache 的新类来避免冗余。AppCache 是处理缓存的应用的核心。把公共代码抽象出来放入 AppCache 可以避免 viewWillAppear:和 viewWillDisappear:中出现冗余代码。

重构这部分代码，使得视图控制器的 viewWillAppear/viewWillDisappear 代码块看起来如下所示。加粗部分显示重构时所做的修改，我会在代码后面解释。

视图控制器的 viewWillAppear:方法中用 AppCache 类缓存数据模型的重构代码片段（MenuItemsViewController.m）

```
-(void) viewWillAppear:(BOOL)animated {
      self.menuItems = [AppCache getCachedMenuItems];
      [self.tableView reloadData];
if([AppCache isMenuItemsStale] || !self.menuItems) {
        [AppDelegate.engine
fetchMenuItemsOnSucceeded:^(NSMutableArray
        *listOfModelBaseObjects) {
        self.menuItems = listOfModelBaseObjects;
        [self.tableView reloadData];
      } onError:^(NSError *engineError) {
```

```
            [UIAlertView showWithError:engineError];
        }];
    }
    [super viewWillAppear:animated];
}
-(void) viewWillDisappear:(BOOL)animated {
    [AppCache cacheMenuItems:self.menuItems];
    [super viewWillDisappear:animated];
}
```

AppCache 类把判断数据是否过期的逻辑从视图控制器中抽象出来了，还把缓存保存的位置也抽象出来了。稍后在本章中我们还会修改 AppCache，再引入一层缓存，内容会保存在内存中。

因为 AppCache 抽象出了缓存的保存位置，我们就不需要为复制粘贴代码来获得应用的缓存目录而操心了。如果应用类似于 iHotelApp，开发者可通过为每个用户创建子目录即可轻松增强缓存数据的安全性。然后我们就可以修改 AppCache 中的辅助方法，现在它返回的是缓存目录，我们可以让它返回当前登录用户的子目录。这样，一个用户缓存的数据就不会被随后登录的用户看到了。

完整的代码可以从本书网站上本章的源代码下载中获取。

缓存版本控制：

我们在上一节中写的 AppCache 类从视图控制器中抽象出了按需缓存。当视图出现和消失时，缓存就在幕后工作。然而，当你更新应用时，模型类可能会发生变化，这意味着之前归档的任何数据将不能恢复到新的模型上。正如之前所讲，对按需缓存来说，数据并没有那么重要，开发者可以删除数据并更新应用。我会展示可以用来在版本升级时删除缓存目录的代码片段。

iOS 中验证模型：

第二个是验证模型，服务器通常会发送一个校验和（Etag）。后续所有从缓存获得资源的请求都应该用这个校验和向服务器**重新验证**资源是否有变化。如果校验和匹配，服务器就返回一个 HTTP 304 Not Modified 的状态码。

IOS 内存缓存：

目前为止，所有 iOS 设备都带有闪存，而闪存有点小问题：它的读写寿命是有限的。尽管这个寿命跟设备的使用寿命比起来很长，但是仍然需要避免过于频繁地读写闪存。在上一个例子中，视图隐藏时是直接缓存到磁盘的，而视图显示时又是直接从磁盘读取的。这种行为会使用户设备的缓存负担很重。为避免这个问题，我们可以再引入一层缓存，利用设备的 RAM 而不是闪存（用 NSMutableDictionary）。在24.2.1节的“实现数据模型缓存”中，我们介绍了创建归档的两种方法：一个是保存到文件，另一个是保存为 NSData 对象。这次会用到第二个方法，我们会得到一个 NSData 指针，将该指针保存到 NSMutableDictionary 中，而不是文件系统里的平面文件。引入内存缓存的另一个好处是，在归档和反归档内容时性能会略有提升。听起来很复杂，实际上并不复杂。本节将介绍如何给 AppCache 类添加一层透明的、位于内存中的缓存。（“透明”是指调用代码，即视图控制器，甚至不知道这层缓存的存在，而且也不需要改动任何代码。）我们还会设计一个 LRU（Least Recently Used，最近最少使用）算法来把缓存的数据保存到磁盘。

以下简单列出了要创建内存缓存需要的步骤。这些步骤将会在下面几节中详细解释。

- 添加变量来存放内存缓存数据。
- 限制内存缓存大小，并且把最近最少使用的项写入文件，然后从内存缓存中删除。RAM 是有限的，达到使用极限就会触发内存警告。收到警告时不释放内存会使应用崩溃。我们当然不希望发生这种事，所以要为内存缓存设置一个最大阈值。当缓存满了以后再添加任何东西时，最近最少使用的对象应该被保存到文件（闪存中）。
- 处理内存警告，并把内存缓存以文件形式写入闪存。
- 当应用关闭、退出，或进入后台时，把内存缓存全部以文件形式写入闪存。

原文链接：http://www.cnblogs.com/qiqibo/p/3520635.html?utm_source=tuicool

Android 学习笔记之 SQLite 基础用法

SQLite 是 Android 中的轻量级的数据库，其基本操作有增、删、查、改。每一种操作都有两个方法，一种是通过 SQL 语句来执行，一种是用 Android 提供的方法。

一、创建数据库（数据库只创建一次）

```
六
4
11
18
25
1
8
```

```
1 public class DBHelper extends SQLiteOpenHelper {
2
3     private static final String DB_NAME = "Test.db";// 数据库名称
4     private static final String TBL_NAME_TEST = "TestTabName"; /
5     // 创建数据库的SQL语句
6     private static final String CREATE_TBL_TEST = "create table
primary key autoincrement,TestNum text,TestName text)";
7     private SQLiteDatabase db;
8
9     /**
10     * 构造函数
11     *
12     * @param context
13     *              上下文
14     * @param name
15     *              数据库名称
16     * @param factory
17     * @param version
18     *              版本号
19     */
20    public DBHelper(Context context) {
21        super(context, DB_NAME, null, 2);
22        // TODO Auto-generated constructor stub
23    }
24
25    // 创建数据库
26    @Override
27    public void onCreate(SQLiteDatabase db) {
28        // TODO Auto-generated method stub
29        this.db = db;
30        // 创建表
31        db.execSQL(TBL_NAME_TEST);
32
33    }
```

二、数据库的操作

1>增（也就是向指定的数据库中插入一条数据）

```
1 /**
2      * 向指定数据库中插入一条数据
3      *
4      * @param values
5      *              ContentValues 键值对， 相当于map
6      * @param tableName
7      *              表名称
8      */
9     public void insert(ContentValues values, String tableName) {
10
11        SQLiteDatabase db = getWritableDatabase();
12        db.insert(tableName, null, values);
13        db.close();
14
15    }
```

2>删（可以删除表中所有数据，也可以指定满足条件的数据）

```
/**
     * 根据ID删除一条数据
     *
     * @param id
     * @param tableName
     */
    public void del(int id, String tableName) {

        if (db == null)
            db = getWritableDatabase();
        db.delete(tableName, "_id=?", new String[] { String.valueOf(id) });

    }

    /**
     * 删除表中所有数据
     *
     * @param tableName
     */
    public void delAll(String tableName) {

        if (db == null)
            db = getWritableDatabase();
        String sql = "Delete from " + tableName;
        try {
            db.execSQL(sql);
        } catch (Exception e) {
            // TODO: handle exception
            System.out.print(e);
        }

    }
```

3>改（更新一条指定的数据）

```java
/**
     * 更新一条数据
     *
     * @param values
     *            要更新的数据
     * @param id
     *            更新的条件
     * @param tableName
     *            更新的表名称
     * @return
     */
    public boolean updataData(ContentValues values, int id, String tableName) {
        boolean bool = false;
        SQLiteDatabase db = getWritableDatabase();
        bool = db.update(tableName, values, "_id=" + id, null) > 0;
        return bool;

    }
```

4>查（查询数据，返回的是游标类型的数据，对它进行读取，打开一个游标，当结束后要关闭游标）

```java
/**
     * 返回表中所有数据
     *
     * @param tableName
     *             表名称
     * @return
     */
    public List<String> quertAll(String tableName) {
        List<String> list = new ArrayList<String>();
        Cursor c = null;
        SQLiteDatabase db = getWritableDatabase();
        c = db.query(tableName, null, null, null, null, null, null);
        // 提取游标中的值
        try {
            for (c.moveToFirst(); !c.isAfterLast(); c.moveToNext()) {
                // 根据列名获取数据
                String TestNum = c.getString(c.getColumnIndex("TestNum"));
                list.add(TestNum);
            }
        } catch (Exception e) {
            // TODO: handle exception
        } finally {
            //关闭游标
            c.close();
        }

        return list;
    }
```

根据条件获取表中的值

```
/**
     * 根据条件查询数据
     *
     * @param where
     *            条件
     * @param tableName
     *            表名称
     * @return   这边我就不获取游标中的值了，同上。
     */
    public Cursor quertToWhere(String where, String tableName) {

        Cursor c = null;
        SQLiteDatabase db = getWritableDatabase();
        try {
            c = db.query(tableName, null, where, null, null, null, null);
        } catch (Exception e) {
            // TODO: handle exception
            String msg = e.toString();
            Log.i("", msg);
        }
        return c;
    }
```

由于我也是个菜鸟，都是比较基础的知识，希望能帮助和我一样菜的同孩们。。。

原文链接：http://www.cnblogs.com/wjdawx/p/3517839.html?utm_source=tuicool

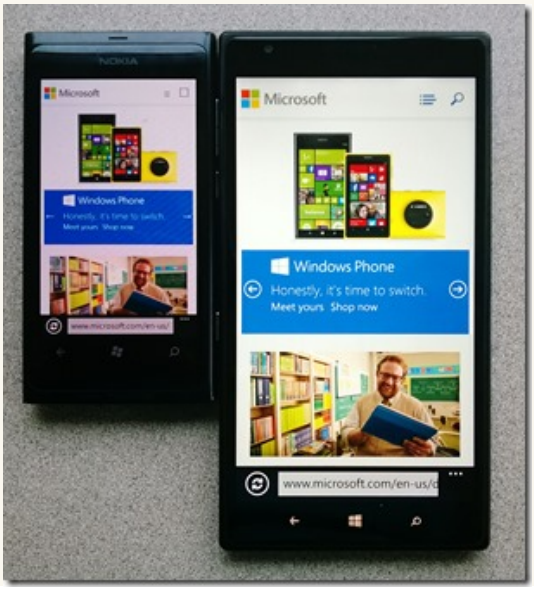
如何充分利用 Windows Phone 高清屏幕

Nokia 最近发布两款6寸大屏手机：Lumia 1520 和 Lumia 1320。为了支持这种设备 WP 升级了操作系统 GDR3 支持了 1080P 的高清分辨率（1520），虽然 GER3 是提供了向下兼容的，当然 GDR3 同时支持一些特性来支持 1080P 高清屏幕。

一下所有的讨论的代码实现都在 这里

Windows Phone 7 开始实现了统一分辨率规范 WVGA（ 800 x 480 ），一般适用与 3.7 - 4.3 寸屏。 在 Windows Phone 8 扩展支持了多种分辨率（3种）WVGA，WXGA（768 x 1280），和 720P（720 x 1280），但是他们支持的物理尺寸多数是在 4 - 4.5寸之间，并且无论应用运行在何种分辨率的机器上我们都是从基础分辨率(800 x 480)进行适配，例如 720P 分辨率，屏幕的宽高比是 16:9，它会从基础分辨率进行1.5倍的放大，但是由于宽高比的原因，实际缩放前的分辨率是 480 x 853，高度多出53个像素来适应 720P 的分辨率。 另外 768 x 1280 和基础分辨率的屏幕宽高比都是 15:9 的所以可以直接进行一个 1.6 的屏幕缩放即可。

随着 Windows Phone 8 的 GDR3 的发布，不仅有一个（1920 x 1080）高清分辨率的加入，同样随之带来一些和大屏是手机兼容性的问题。例如一个相同页面在 3.7 寸屏幕上显示和在一个 6寸设备上显示的不同效果，和用户体验。



理论上讲，Windows Phone 也许会运行在更大（7“）屏的设备上，所以我们在需要的时候充分的利用屏幕，至少要在应用中知道我们当前是在何种分辨率的设备中，但是如果我们什么都不做的情况下系统会帮我们进行一个 720P 的应用适配（为了兼容现有应用），也就是从480 x 853进行缩放，但是他不是简单的从 720P 缩放到 1080P，系统会从新渲染所有控件显示到 1080P 设备上。

所以我们现有的 App 可以继续运行在 1080P 的设备上且不会收到影响，但是针对一个新应用我们怎么做才能更好的适配一个1080P 的设备呢？例如：检测到当前设备是支持1080P 的时候我们播放的视频进行一个适配，所以在 GDR3 SDK 支持一对新的可侦测属性参数。PhysicalScreenResolution 和 RawDpiX

```
namespace ScreenSizeBlogPostApp
{
  2 references
  public partial class BasicInfo : PhoneApplicationPage
  {
    0 references
    public BasicInfo()
    {
      InitializeComponent();
    }

    1 reference
    string GetBasicScreenInfo()
    {
      var width = App.Current.Host.Content.ActualWidth;
      var height = App.Current.Host.Content.ActualHeight;
      var scaleFactor = (double)App.Current.Host.Content.ScaleFactor / 100d;

      return String.Format("{0} x {1}; {2:0.0} scale factor", width, height, scaleFactor);
    }

    1 reference
    string GetExtendedScreenInfo()
    {
      object temp;
      if (!DeviceExtendedProperties.TryGetValue("PhysicalScreenResolution", out temp))
        return "not available, sorry";

      var screenResolution = (Size)temp;

      // Can query for RawDpiY as well, but it will be the same value
      if (!DeviceExtendedProperties.TryGetValue("RawDpiX", out temp) || (double)temp == 0d)
        return "not available, sorry";

      var dpi = (double)temp;
      var screenDiagonal = Math.Sqrt(Math.Pow(screenResolution.Width / dpi, 2) +
                Math.Pow(screenResolution.Height / dpi, 2));

      var width = App.Current.Host.Content.ActualWidth;

      return String.Format("{0} x {1}; {2:0.0#} raw scale; {3:0.0}\"", screenResolution.Width, screenResolution.Height, screenResolution.Width / width, screenDiagonal);
    }

    0 references
    void GetScreenInfoClick(object sender, RoutedEventArgs e)
    {
      basicScreenInfoOutput.Text = GetBasicScreenInfo();
      extendedScreenInfoOutput.Text = GetExtendedScreenInfo();
    }
  }
}
```

```
public partial class BasicInfo : PhoneApplicationPage
  {
    public BasicInfo()
```

```
        {
          InitializeComponent();
        }
        string GetBasicScreenInfo()
        {
          var width = App.Current.Host.Content.ActualWidth;
          var height = App.Current.Host.Content.ActualHeight;
          var scaleFactor =
(double)App.Current.Host.Content.ScaleFactor / 100d;
          return String.Format("{0} x {1}; {2:0.0} scale factor", width,
height, scaleFactor);
        }
        string GetExtendedScreenInfo()
        {
          object temp;
          if
(!DeviceExtendedProperties.TryGetValue("PhysicalScreenResolution",
out temp))
            return "not available, sorry";
          var screenResolution = (Size)temp;
          // Can query for RawDpiY as well, but it will be the same value
          if (!DeviceExtendedProperties.TryGetValue("RawDpiX", out
temp) || (double)temp == 0d)
            return "not available, sorry";
          var dpi = (double)temp;
          var screenDiagonal =
Math.Sqrt(Math.Pow(screenResolution.Width / dpi, 2) +
                        Math.Pow(screenResolution.Height / dpi, 2));
```

```
            var width = App.Current.Host.Content.ActualWidth;
            return String.Format("{0} x {1}; {2:0.0#} raw scale;
{3:0.0}\"", screenResolution.Width, screenResolution.Height,
screenResolution.Width / width, screenDiagonal);
        }
        void GetScreenInfoClick(object sender, RoutedEventArgs e)
        {
          basicScreenInfoOutput.Text = GetBasicScreenInfo();
          extendedScreenInfoOutput.Text = GetExtendedScreenInfo();
        }
      }
    }
```

例如 在 GDR3 Lumia 920 中运行以上代码的效果：

在 720P 模拟器中运行的效果（没有 GDR3的更新）：

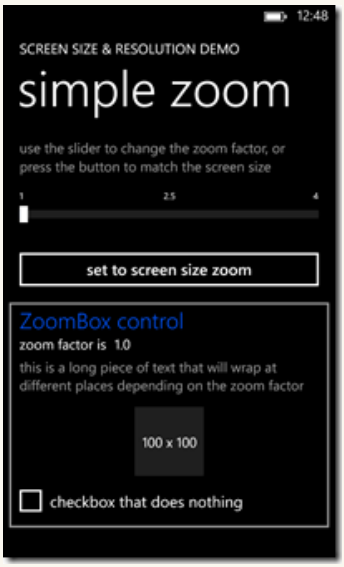
在6寸设备上运行效果，（基础信息和 720P 相同）：

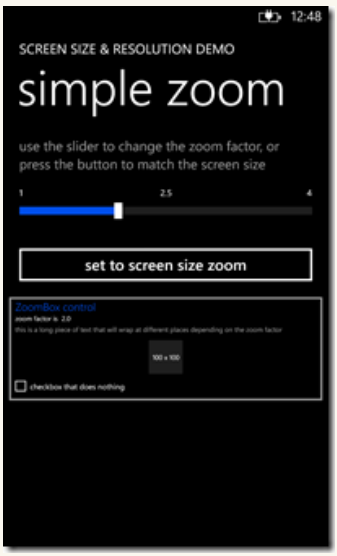
但是我们可以获取到一些实际的数值来帮助开发者更好的适配自己的应用。

ZoomBox control

在上面提到的代码示例中有一个叫做 ZoomBox 的控件，类似于 ViewBox，但是 ViewBox 可以让内部控件比他本身更大，但是你可以结合 DisplayInformationEx 使用，可以更好的适应屏幕。（这里使用了一个滑块 slider 来调整控件区域的大小）

细心的同学可以看到随着滑块的滑动 文字/图片/控件 大小的变化（文字可以自动换行），实际上我们可以在这里使用任何缩放的数值，来根据实际的屏幕尺寸。





Rendering size - aware content

正如前面说到的不论我们在何种分辨率下进行开发我们在 XAML 中都是用800*480 这基础分辨率下进行的设计。但是会有这样的一种情况出现，一个4.5寸设备分辨率是480pixels,如果相同的分辨率到一个6寸的设备上那就相当于每一个像素都放大了33%，实际大小就相当于原有的638px 的大小。DisplayInformationEx 可以计算这些数值并且配合缩放来使得控件大小看上去一致。

名称解析：

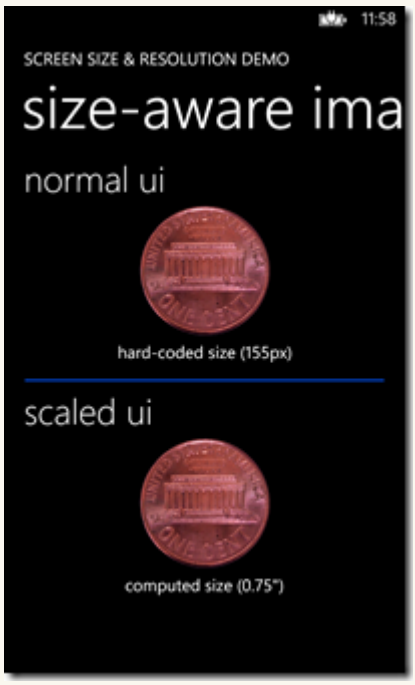
Raw Pixels ： 原始像素值 = 实际硬件上的像素数量，例如1080P 的设备横向拥有1080个像素点。

Host Pixels：设计/基础 像素 = 在我们设计应用的时候 xaml runtime 提供的像素数，例如所以设备的基数像素点都是480pixels。

View Pixels：显示像素 = 通过 DisplayInformationEx 计算适配后 ZoomBox 显示出来的像素数，同时和机器的屏幕物理尺寸相关。

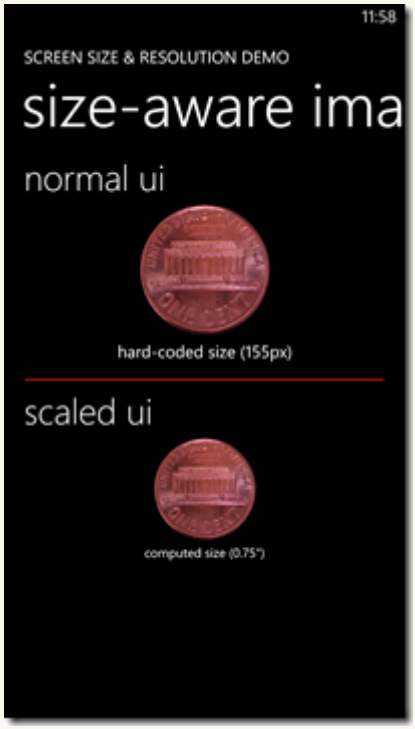
有一种情况 例如 Nokia Lumia 820（4.3”） 它的 Raw Pixels,Host Pixels,以及 View Pixels 都是相同的。但是在更高分辨率的机器上就会不同了。

示例中的代码是使用 ZoomBox 在 Lumia 920中显示一个硬币保持硬币的直径在 0.75 “。



在上面的这张图中看到 通过 ZoomBox 的修正，让 155px 的 硬币在1.6倍放大的屏幕上（4.5”）表现出与原始大小相符的图片。

另外看一下在 6寸屏幕上显示的效果：



对比一下 lumia 920 和 6寸设备显示的效果。

ZoomBox 和 DisplayInformationEx 的使用方法

原文： http://www.cnblogs.com/sonic1abc/p/3517575.html?utm_source=tuicool

【cocos2d-x 手游研发----博彩大转盘】

博彩大转盘，转盘抽奖的小系统，这是一个很有意思的游戏模块，游戏中增加这样一些趣味的小模块，我会附上源码；

会增进玩家的粘性，每天都想来抽两把试试手气；

我做的这个是个矩形风格的转盘，不是那种圆形的转盘，但是原理是相差不多的；

首先准备一些素材，如：奖品，转盘格子背景，开始按钮等等....

接下来，我想把这个转盘系统单独做在一个 class 文件夹中，以后可插拔的方便接入任何游戏，建了一个文件夹 ZhuanPanSystem；

说一下大概的制作思路，首先是需要格子，来组成一个矩形矩阵，长和宽根据自己需求自己去设置，中间有一个按钮，点了之后，格子

就会变背景，并且循环跑动在矩形格子上，最终根据加速度从快到慢，减速下来停在哪个格子上，便获取到该格子上的奖励；

ok 思路一定，看下代码如何实现；

格子：

```
//创建一个矩阵格子阵
    boxgezi = CCArray::create();
    int bid = 0;
    for (int i = 0; i < 4; i++)
```

```
        {
            for (int j = 0; j <=5; j++)
            {
                Zp_BoxData* thisbox = new Zp_BoxData();
                thisbox->set_boxid(bid);
                thisbox->set_boxReward(getReward(bid%6));
                if(bid==4)
                {
                    thisbox->set_xuanzhong(true);
                }
                else            {
                    thisbox->set_xuanzhong(false);
                }
                thisbox->set_tag(bid);
                switch (i)
                {
                case 0:
                    thisbox->set_point(ccp(55+gezi_w*j,allbd_h-30));
                    boxgezi->addObject(thisbox);
                    break;
                case 1:
                    if(j<=2)
{               thisbox->set_point(ccp(55+gezi_w*5,allbd_h-30-gezi_h-g
ezi_h*j));
                        boxgezi->addObject(thisbox);
                    }
                    break;
                case 2:
                    thisbox->set_point(ccp(55+gezi_w*5-gezi_w*j,30));
```

```
                    boxgezi->addObject(thisbox);
                    break;
                case 3:
                    if(j<=2)
                    {
                    thisbox->set_point(ccp(55,30+gezi_h+gezi_h*j));
                        boxgezi->addObject(thisbox);
                    }
                    break;
                default:
                    break;
                }
                bid++;
            }
        }
        vector<int> maua ;
        for (int i = 0; i < boxgezi->count(); i++)
        {
            Zp_BoxData*                    bdata                    =
(Zp_BoxData*)boxgezi->objectAtIndex(i);
            Gzi* gz = new Gzi(allbd,bdata);
            maua.push_back(bdata->get_boxid());
        }
```

以上就是利用 Gzi 类创建出来的一个矩形范围的矩阵转盘，有了矩形转盘，还需要游戏中的点击开始以后，循环转动的效果；

用了一个递归去循环去跑一个加速度的效果的方法：

```
    void TurntableSystem::runTurntableGet(float time)
    {
        this->schedule(schedule_selector(TurntableSystem::runAct),
```

```
time);
    }
    void TurntableSystem::runAct(float time)
    {
        vector<int>                         gezi_l                          =
GlobalInfo::getInstance()->get_gizilist();
        //做事儿
        if(gezi_l.size()>0)
        {
            if(fnum>gezi_l.size()-1)
            {
                fnum = 0;
            }
            int bid = gezi_l.at(fnum);
            changeBox(bid,true);
            //再把上一个变回来
            int lastnum = fnum-1;
            if(lastnum<0)
            {
                lastnum=gezi_l.size()-1;
            }
            int lastid = gezi_l.at(lastnum);
            changeBox(lastid,false);
            fnum++;
        }
        runnum++;
        this->unschedule(schedule_selector(TurntableSystem::runAct));
        CCLOG("------%f----times=%d-",time,runnum);
        if(runnum<25)
```

```
        {
            float nexttime =  time+runnum*0.01f;
            if(nexttime>=1.5f)
            {
                nexttime=1.5f;
            }
this->schedule(schedule_selector(TurntableSystem::runAct),nexttime);
        }
    }
```

这边我是启动了一个定时器去实现这个递归加速的方法，里面的25目前是固定的跑25格必定停下!!!

以下就牵扯到随机数概率获取奖品的问题了，那么根据咱们策划给的方案，每个格子的概率对应的格子数，和步数

去 set 这个值就可以了，剩下的工作就很简单了，只需增加随机概率就可以了；

下面我帖一下跑起来的效果图：

开始后循环跑动；

原文：http://www.cnblogs.com/zisou/p/cocos2d-xZhuanpan.html?utm_source=tuicool

回顾 2013：HBase 的提升与挑战

2013年马上就要过去了，总结下这一年 HBase 在这么一年中发生的主要变化。影响最大的事件就是 HBase 0.96的发布，代码结构已经按照模块化发布了，而且提供了许多大家迫切需求的特点。这些特点大多在 Yahoo!/Facebook/淘宝/小米等公司内部的集群中跑了挺长时间了，可以算是比较稳定可用了。

1. Compaction 优化

HBase 的 Compaction 是长期以来广受诟病的一个特性，很多人吐槽 HBase 也是因为这个特征。不过我们不能因为 HBase 有这样一个缺点就把它一棒子打死，更多的还是希望能够驯服它，能够使得它适应自己的应用场景。根据业务负载类型调整 Compaction 的类型和参数，一般在业务高峰时候禁掉 Major Compaction。在0.96中 HBase 社区为了提供更多的 Compaction 的策略适用于不同的应用场景，采用了插件式的架构。同时改进了 HBase 在 RegionServer 端的存储管理，原来是直接 Region->Store->StoreFile，现在为了支持更加灵活多样的管理 StoreFile 和 Compact 的策略，RS 端采用了 StoreEngine 的结构。一个 StoreEngine 涉及到 StoreFlusher、CompactionPolicy、Compactor、StoreFileManager。不指定的话默认是 DefaultStoreEngine，四个组件分别是 DefaultStoreFlusher，ExploringCompactionPolicy、DefaultCompactor、DefaultStoreFileManager。可以看出在0.96版之后，默认的 Compaction 算法从 RatioBasedCompactionPolicy 改为了 ExploringCompactionPolicy。为什么要这么改，首先从 Compaction 的优化目标来看：compaction is about trading some disk IO now for fewer seeks later，也就是 Compaction 的优化目标是执行 Compaction 操作能合并越多的文件越好，如果合并同样多的文件产生的 IO 越小越好，这样 select 出来的列表才是最优的。

主要不同在于：

- RatioBasedCompactionPolicy 是简单的从头到尾遍历 StoreFile 列表，遇到一个符合 Ratio 条件的序列就选定执行 Compaction。对于典型的不断 flush memstore 形成 StoreFile 的场景是合适的，但是对于 bulk-loaded 是不合适的，

会陷入局部最优。

- 而 ExploringCompactionPolicy 则是从头到尾遍历的同时记录下当前最优，然后从中选择一个全局最优列表。

关于这两个算法的逻辑可以在代码中参考对应的 applyCompactionPolicy()函数。其他 CompactionPolicy 的研究和开发也非常活跃，例如 Tier-based compaction（HBASE-6371，来自 Facebook）和 stripe compaction（HBASE-7667）

吐槽：HBase Compaction 为什么会问题这么多，我感觉缺少了一个整体的 IO 负载的反馈和调度机制。因为 Compaction 是从 HDFS 读数据，然后再写到 HDFS 中，和其他 HDFS 上的负载一样在抢占 IO 资源。如果能有个 IO 资源管理和调度的机制，在 HDFS 负载轻的时候执行 Compaction，在负载重的时候不要执行。而且这个问题在 Hadoop/HDFS 里同样存在，Hadoop 的资源管理目前只针对 CPU/Memory 的资源管理，没有对 IO 的资源管理，会导致有些 Job 受自己程序 bug 的影响可能会写大量的数据到 HDFS，会严重影响其他正常 Job 的读写性能。

更多内容：HBase Compaction、HBase 2013 Compaction 提升。

2. Mean Time To Recovery/MTTR 优化

目前 HBase 对外提供服务，Region Server 是单点。如果某台 RS 挂掉，那么直到该 RS 上的所有 Region 被重新分配到其他 RS 上之前，这些 Region 是的数据是无法访问的。对这个过程的改进主要包括：

- HBASE-5844 和 HBASE-5926：删除 zookeeper 上 Region Server/Master 对应的 znode，这样就省的等到 znode 30s 超时才发现对应的 RS/Master 挂了。
- HBASE-7006： Distributed Log Replay，就是直接从 HDFS 上读取宕机的 WAL 日志，直接向新分配的 RS 进行 Log Replay，而不是创建临时文件 recovered.edits 然后再进行 Log Replay
- HBASE-7213/8631： HBase 的 META 表中所有的 Region 所在的 Region Server 将会有两个 WAL，一个是普通的，一个专门给 META 表对应的 Region 用。这样在进行 recovery 的时候可以先恢复 META 表。

3. Bucket Cache （L2 cache on HBase）

HBase 上 Regionserver 的内存分为两个部分，一部分作为 Memstore，主要用来写；另外一部分作为 BlockCache，主要用于读。Block 的 cache 命中率对

HBase 的读性能影响十分大。目前默认的是 LruBlockCache，直接使用 JVM 的 HashMap 来管理 BlockCache，会有 Heap 碎片和 Full GC 的问题。

HBASE-7404引入 Bucket Cache 的概念可以放在内存中，也可以放在像 SSD 这样的适合高速随机读的外存储设备上，这样使得缓存的空间可以非常大，可以显著提高 HBase 读性能。Bucket Cache 的本质是让 HBase 自己来管理内存资源而不是让 Java 的 GC 来管理，这个特点也是 HBase 自从诞生以来一直在激烈讨论的问题。

4. Java GC 改进

MemStore-Local Allocation Buffers 通过预先分配内存块的方式解决了因为内存碎片造成的 Full GC 问题，但是对于频繁更新操作的时候，MemStore 被 flush 到文件系统时没有 reference 的 chunk 还是会触发很多的 Young GC。所以 HBase-8163提出了 MemStoreChunkPool 的概念，也就是由 HBase 来管理一个 ChunkPool 用来存放 chunk，不再依赖 JVM 的 GC。这个 ticket 的本质也是由 HBase 进程来管理内存分配和重分配，不再依赖于 Java GC。

5. HBase 的企业级数据库特性（Secondary Index、Join 和 Transaction）

谈到 HBase 的企业级数据库特性，首先想到的就是 Secondary Index、Join、Transaction。不过目前这些功能的实现都是通过外围项目的形式提供的。

华为的 hindex 是目前看到的最好的 Secondary Index 的实现方式。主要思想是建立 Index Table，而且这个 Index Table 的 Region 分布跟主表是一致的，也就是说主表中某一 Region 对应的 Index Table 对应的 Region 是在同一个 RS 上的。而且这个索引表是禁止自动或者手动出发 split 的，只有主表发生了 split 才会触发索引表的 split。

这个是怎么做到的呢？本质上 Index Table 也是一个 HBase 的表，那么也只有一个 RowKey 是可以索引的。这个索引表的 RowKey 设计就比较重要了，索引表的 RowKey=主表 Region 开始的 RowKey+索引名（因为一个主表可能有多个索引，都放在同一个索引表中）+需要索引的列值+主表 RowKey。这样的索引表的 RowKey 设计就可以保证索引表和主表对应的 Region 是在同一台 RS 上，可以省查询过程中的 RPC。每次 Insert 数据的时候，通过 Coprocessor 顺便插入到索引表中。每次按照二级索引列 Scan 数据的时候，先通过 Coprocessor 从索引表中获取对

应的主表的 RowKey 然后就行 Scan。在性能上看，查询的性能获得了极大提升，插入性能下降了10%左右。

Phoenix 也通过另外建一张表的方式实现二级索引

Phoenix 也实现了一大一小两个表的 Join 操作。还是老办法把小表 broadcast 到所有的 RS，然后通过 coprocessor 来做 hash join，最后汇总。感觉有点画蛇添足，毕竟 HBase 设计的初衷就是用大表数据冗余来尽量避免 Join 操作的。现在又来支持 Join，不知道 Salesfore 的什么业务需求这个场景。

关于 Transaction 的支持，目前最受关注的还就是 Yahoo!的 Omid。不过貌似大家对这个特性的热情还不是特别高。

6. PrefixTreeCompression

由于 HBase 的 KeyValue 存储是按照 Row/Family/Qualifier/TimeStamp/Value 的形式存储的，Row/Family/Qualifier 这些相当于前缀，如果每一行都按照原始数据进行存储会导致占据存储空间比较大。HBase 0.94版本就已经引入了 DataBlock Encode 的概念(HBASE-4218),将重复的 Row/Family/Qualifier 按照顺序进行压缩存储，提高内存利用率，支持四种压缩方式 FAST_DIFF\PREFIX\PREFIX_TRIE\DIFF。但是这个特性也仅仅是通过 delta encoding/compression 降低了内存占用率，对数据查询效率没有提升，甚至会带来压缩/解压缩对 CPU 资源占用的情况。

HBASE-4676：PrefixTreeCompression 是把重复的 Row/Family/Qualifier 按照 Prefix Tree 的形式进行压缩存储的，可以在解析时生成前缀树，并且树节点的儿子是排序的，所以从 DataBlock 中查询数据的效率可以超过二分查找。（PREFIX_TREE 压缩的初步探究及测试）

7. 其他变化

- HBASE-5305：为了更好的跨版本的兼容性，引进了 Protocol Buffer 作为序列化/反序列化引擎使用到 RPC 中（此前 Hadoop 的 RPC 也全部用 PB 重写了）。因为随着 HBase Server 的不断升级，有些 Client 的版本可能还比较旧，所以需要 RPC 在新旧版本之间兼容。
- HBASE-6055 HBASE-7290：HBase Table Snapshot。创建 snaphost 对 HBase 集群没有性能影响，只是生成了 snaphost 对应的 metadata 而不会去拷贝数据。

用户可以通过创建 snaphost 实现 backup 和 disaster recovery，例如用户在创建一个 snaphost 之后可能会误操作导致一些表出现了问题，这样我们可以选择回滚到创建 snaphost 的那个阶段而不会导致数据全都不可用。也可以定期创建 snapshot 然后拷贝到其他集群用于定时的离线处理。

- HBASE-8015： 在0.96中，ROOT 表已经改名为 hbase:namespace，META 则是 hbase:meta。而且 hbase:namespace 是存在 Zookeeper 上的。这个 namespace 类似于 RDBMS 里的 database 的概念，可以更好的做权限管理和安全控制。HBase 中 table 的 META 信息也是作为一种 Region 存放在 Region Server 上的，那么 META 表的 Region 和其他普通 Region 就会产生明显的资源竞争。为了改善 META Region 的性能，360的 HBase 中提出了专属 MetaServer，在这个 Region Server 上只存放 META Region

- HBASE-5229：同一个 Region 内的跨行事务。一次操作中涉及到同一个 Region 中的所有写操作在获取到的相关 Row 的所有行锁（按照 RowKey 的顺序依次取行锁，防止死锁）之后事务执行。

- HBASE-4811:Reverse Scan。过去被问到说如何反向查找 HBase 中的数据，常常被答道再建一张反向存储的表,而且 LevelDB 和 Cassandra 都支持反向扫描。HBase 反向扫描比正向扫描性能下降30%，这个和 LevelDB 是差不多的。

- Hoya — HBase on YARN。可以在一个 YARN 集群上部署多个不同版本、不同配置的 HBase 实例，可以参考 GitHub。

展望2014年，HBase 即将 release 1.0版本，更好的支持 multi-tenancy，支持 Cell 级别的 ACL 控制。

8. 总结

- Cloudera/Hortonworks/Yahoo!/Facebook 的人从系统和性能等多方面关注

- Salesfore/huawei 的人貌似更关注企业级特性，毕竟他们面对的客户都是电信、金融、证券等高帅富行业

- 来自国内的阿里巴巴/小米/360等公司更加关注系统性能、稳定性和运维相关的话题。国内互联网行业用 HBase 更加关注的是如何解决业务问题。

- 越来越多的公司把它们的 HBase 集群构建在云上，例如 Pinterest 所有的

HBase 集群都是在 AWS 上，国外的 start up 环境太好了，有了 AWS 自己根本不用花费太多的资源在基础设施上。

- 传统的 HBase 应用是在线存储，实时数据读取服务。例如支付宝用 HBase 存放用户的历史交易信息服务用户查询，中国联通也使用 HBase 存储用户的上网历史记录信息用于用户的实时查询需求。现在 HBase 也向实时数据挖掘的应用场景中发展，例如 wibidata 公司开源的 kiji 能够在 HBase 上轻松构建实时推荐引擎、实时用户分层和实时欺诈监控。

原文链接：http://www.csdn.net/article/2014-01-15/2818147-hbase-in-2013?utm_source=tuicool

memcached（十七）协议命令格式

memcached 的管理使用的是 telnet

登录服务器 telnet 127.0.0.1 11211

<command name> <key> <flags> <exptime> <bytes> [noreply]\r\n

cas <key> <flags> <exptime> <bytes> <cas unique> [noreply]\r\n

<command name> ： "set", "add", "replace", "append" or "prepend"

set：“存储这个数据”，一般是更新已有的缓存，也可以用于新增。

add：新增缓存，缓存中不存在新增的 KEY。

replace：替换现有的缓存，缓存中一定已经存储 KEY

append：在现有的缓存数据后添加缓存数据。

prepend：在现有的缓存数据前添加缓存数据

cas：check and set 操作，存储缓存，前提是在 check 后没有其它人修改过数据，用于多客户端同时设置相同的 KEY 时的原子操作。

<key>：缓存的 KEY

<flags>：最开始是 16 位的无符号整数，现在的版本一般是 32 位。用户客户端存储自定义标记数据。

<exptime>：缓存过期时间。0 表示永不过期，可以是 Unix time 或当前服务器时间的偏移量（秒为单位），如果你想设置当前时间后 1 分钟过期，则此参数为 60。

<bytes>：缓存数据的长度

<cas unique>：unique 64-bit value of an existing entry，cas 操作的时候回传的值，用于服务器端判断缓存是否改变。

[noreply]：服务器不响应处理结果。

<data block>\r\n： 缓存数据块，\r\n 结束

"STORED\r\n"：表示存储成功

"NOT_STORED\r\n"：表示未存储，但并不是错误。如：对已经有的 KEY 使用 add

"EXISTS\r\n":表示使用 cas 命令设置数据未成功，在你最后一次获取数据后，数据已经被其它人修改。

"NOT_FOUND\r\n"：表示使用 cas 存储数据时候，key 不存储

原文链接：http://phl.iteye.com/blog/2005460?utm_source=tuicool

nginx 大流量负载调优

lnmp 已经成为比较流行的网站服务器端技术配备。越来越多的人开始不满足于能使用 nginx，更多人开始关注如何能优化 nginx 的处理能力。

使用 nginx 的目的就是为了提高并发处理能力,但是看到有部分人本机部署 lanmp,在同一台机器上使用 nginx 方向代理 apache,就有种脱裤子放屁的感觉。

在 window 下运行 nginx，还要跑出好的效果，同样是个伪命题，windows 下的 select 模型注定 nginx 效率不会太高。

最近看了篇英文文章，结合自己理解，写给大家看看吧。

优化 nginx 包括两方面：

1.是自己重写 nginx 代码(比如 tengine)、本身 nginx 的代码已经足够优秀，如果不是每秒几千的请求，就忽略这个部分吧。

2.另一个就是和优化 nginx 的配置，这是中小型网站可以重点优化的部分。

nginx 的配置文件是一种声明式定义，控制 nginx 的每一个细节。

所谓负载调优，就是提高单台机器处理效率，降低单台机器的负载。

为了提高单台机器的处理效率，cpu 的处理速度是足够快的，我们能解决的就是降低磁盘 I/O、网络 I/O，减少内存使用。

降低单台机器的负载我们能做的就是负载均衡，把流量打到多台机器处理。

nginx 推荐优化内容：

1.**open files 数量优化**

ulimit -a 查看系统参数

其中

open files (-n) 1024

表示系统同时最多能打开的文件数，linux 下的所有设备都可以认为是文件，包括网络连接，如果同时超过1024个连接，那么 nginx 的日志就会报“24: Too many open files”

多以优化的第一步就是设置 open files 为 ulimit

修改/etc/profile,增加

ulimit -n 65535

2.**Worker Processes 数量优化**

通常来说设置一个 cpu 核心对应一个 worker processer，最多不超过4个，提高 worker process 的值是为了提高计算能力，但一般在越到 cpu 瓶颈前，你会遇到别的瓶颈(如网络问题)。

只有当你要处理大量静态文件的磁盘 I/O 时，worker 进程是单线程的，所以这个读取文件的阻塞 IO 会降低 CPU 的处理速度,这是可以增加 worker 进程数量，其它情况是不需要的。

3.worker 进程连接数优化(Worker Connections)

默认情况下这个值是 worker_connections 1024，也就是说考虑到 keep-alive 超时65秒，每个浏览器平均消耗两个链接(chrome 会同时打开多个连接来提到加

载速度)。

那么默认情况下 nginx 平均每秒能处理1024/65/2=8，那么8*86440=64w，差不多相当于每天有60万 ip。

多以普通网站默认值就可以了，如果你的流量一直提升，可以考虑增加这个值为2048或者更高。

3. **CPU Affinity**

用来设置 worker 进程使用哪个 cpu 核心处理请求并且一直使用这个 cpu 核心。如果你不知道 cpu 调度，最好别碰这个，操作系统比你更懂如何调度。

4. **Keep Alive**

Keep alive 没有数据传输的情况下保持客户端和服务端的连接，也就是保持空连接一段时间，避免重现建立链接的时间消耗。nginx 处理空连接的效率非常高，1万个空连接大约消耗2.5M 内存。如果流量非常大的网站，减少建立连接的时间开销是非常客观的。keep alive 的值设置在10-20s 之间比较合理。

5. **tcp_nodelay 和 tcp_nopush 优化**

这两个指令影响 nginx 的底层网络，它们决定操作系统如何处理网络层 buffer 和什么时候把 buffer 内容刷新给终端用户。如果你不懂，就可以保持这两个指令默认不变，对 nginx 性能影响不明显。

6. **access 日志优化**

默认情况下，access 日志会记录所有请求到日志文件，写操作会增加 IO 操作，如果不需要统计信息，可以使用百度统计或者 cnzz 统计，完全可以关闭日志，来减少磁盘写，或者写入内存文件，提高 IO 效率。

7. **Error 日志优化**

错误日志会记录运行中的错误，如果设置的太低，会记录的信息太多，会产生大量 IO，推荐设置为 warn，这样可以记录大部分信息，而不会有太多 IO

8. **Open File Cache**

nginx 会读文件系统的许多文件，如果这些文件的描述符能够缓存起来，那么会提高处理效率。详见 http://wiki.nginx.org/HttpCoreModule#open_file_cache

9. **Buffers size 优化**

buffer 的大小是你需要调优最重要参数。如果 buffer size 太小就会到导致 nginx 使用临时文件存储 response，这会引起磁盘读写 IO，流量越大问题越明显。

client_body_buffer_size 处理客户端请求体 buffer 大小。用来处理 POST 提交数据，上传文件等。client_body_buffer_size 需要足够大以容纳如果需要上传 POST 数据。

fastcgi_buffers，proxy_buffers 处理后端（PHP, Apache）响应。如果这个 buffer 不够大，同样会引起磁盘都系 IO。需要注意的是它们有一个上限值，这个上限值受 fastcgi_max_temp_file_size 、 proxy_max_temp_file_size 控制。

10. **磁盘 IO**

如果能把数据全放到内存，不使用磁盘就可以完全去掉磁盘 IO。 默认情况下操作系统也会缓存频繁访问的数据以降低 IO。所以预算足够的情况加，加大内存。

11. **网络 IO**

假设我们没有了磁盘 IO，所有数据都在内存，那么我们的读 IO 大概有3-6gbps。这种情况下，如果你网络差，一样会很慢。所以尽可能提高网络带宽，压缩传输数据。

网络带宽买你能买的起的最大带宽，nginx 的 gzip 模块可以用来压缩传输数据，通常 gzip_comp_level 设为 4-5，再高就是浪费 cpu 了。同时也可以采用 css，js 压缩技术，当然这些技术就与 nginx 优化无关了。。

绝招

如果你还想提高 nginx 处理能力，只能祭出大杀器了。别优化了，加机器吧。一点点优化是没有用的，不如扩展机器来的快些。

ps

说道系统的扩展性通常有 scale、和 extension，区别是前者是数量上扩展，后者是功能上扩展。

原文链接：http://www.nginx.cn/2212.html?utm_source=tuicool

12306 的技术革命

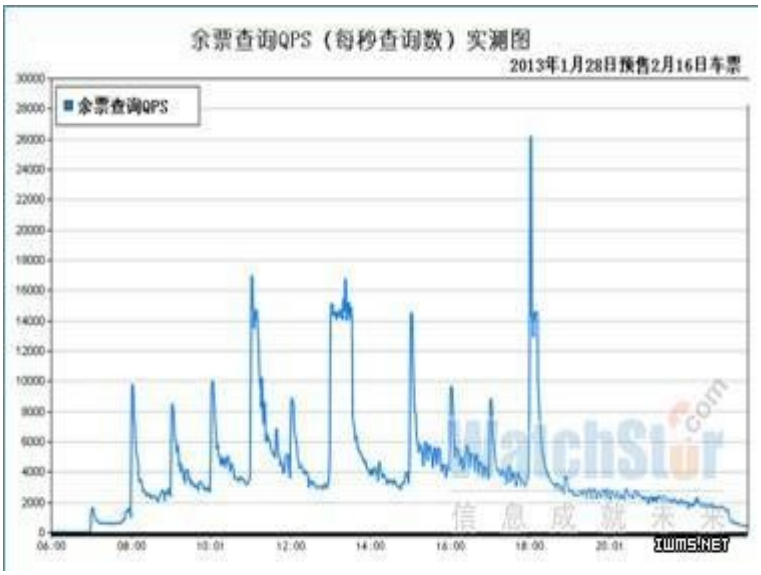
“通过技术改造解决了困扰我们多时的尖峰高流量并发问题，让全国人民不再因为技术原因而抱怨，我们终于舒了一口气。Pivotal GemFire 分布式集群内存数据技术对整个技术改造发挥了关键的作用。同时，感谢 Pivotal 公司及其实施方项目团队的努力，在技术开改造过程中确保旧系统顺畅运行、旧系统到新系统平滑迁移，快速实现新系统的上线。”

——中国铁道科学研究院电子计算技术研究所副所长朱建生

背景和需求

中国铁路客户服务中心网站(www.12306.cn)是世界规模最大的实时交易系统之一，媲美 Amazon.com，节假日尤其是春节的访问高峰，网站压力巨大。据统计， 在2012年初的春运高峰期间，每天有2000万人访问该网站，日点击量最高达到14亿。大量同时涌入的网络访问造成12306几近瘫痪。 中国铁道科学院电子计算技术研究所作为12306互联网购票系统的承建单位，急需寻求办法解决问题。

成功解决：速度提高75倍以上



2012年3月开始，铁路总公司(原铁道部)开始调研、改造12306。2012年6月选择了 Pivotal GemFire 分布式内存计算平台(Distributed In-memory computing)改造12306，由铁科院项目小组负责人王明哲主任和资拓宏宇(IISI)

信息科技有限公司在铁科院主管朱建生所长领导下提供技术实施。一期先改造12306的主要瓶颈——余票查询系统。9月份完成代码改造，系统上线。2012年国庆，又是网上订票高峰期间，大家可以显着发现，可以登录12306，虽然还是很难订票，但是查询余票很快。2012年10月份，二期用 GemFire 改造订单查询系统（客户查询自己的订单记录）。2013年春节，又是网上订票高峰期间，大家可以显着发现，可以登录12306，虽然还是很难订票，但是查询余票很快，而且查询自己的订票和下订单也很快。

根据系统运行数据记录，技术改造之后，在只采用10几台 X86服务器实现了以前数十台小型机的余票计算和查询能力，单次查询的最长时间从之前的15秒左右下降到0.2秒以下，缩短了75倍以上。2012年春运的极端高流量并发情况下，系统几近瘫痪。而在改造之后，支持每秒上万次的并发查询，高峰期间达到2.6万个查询/秒吞吐量，整个系统效率显着提高。如上图所示。

订单查询系统改造，在改造之前的系统运行模式下，每秒只能支持300-400个查询/秒的吞吐量，高流量的并发查询只能通过分库来实现。改造之后，可以实现高达上万个查询/秒的吞吐量，而且查询速度可以保障在20毫秒左右。

新的技术架构可以按需弹性动态扩展，并发量增加时，还可以通过动态增加X86服务器来应对，保持毫秒级的响应时间。

梦里寻它：技术革命一步跨越三代

12306能够取得这样翻天覆地的效果，靠技术上的小修小补是不可能的，必须有全新的思路，能够给性能提升带来杠杆式的作用。12306发现 GemFire 分布式内存数据平台就是这样一种技术。

GemFire 分布式内存数据平台的技术原理如上图所示：通过云计算平台虚拟化技术，将若干 X86服务器的内存集中起来，组成最高可达数十 TB 的内存资源池，将全部数据加载到内存中，进行内存计算。计算过程本身不需要读写磁盘，只是定期将数据同步或异步方式写到磁盘。GemFire 在分布式集群中保存了多份数据，任何一台机器故障，其它机器上还有备份数据，因此通常不用担心数据丢失，而且有磁盘数据作为备份。GemFire 支持把内存数据持久化到各种传统的关系数据库、Hadoop 库和其它文件系统中。

大家知道，当前计算架构的瓶颈在存储，处理器的速度按照摩尔定律翻番增长，而磁盘存储的速度增长很缓慢，由此造成巨大高达10万倍的差距(如上图)。这样就很好理解 GemFire 为什么能够大幅提高系统性能了。

按照计算与存储的关系，我们可以将计算架构分为四代：

第一代，基于磁盘的单一系统：计算过程中需要从磁盘读取数据。小型机、大型机是其中的佼佼者，将单一系统的性能做到极致。

第二代，基于磁盘的分布式集群系统：计算过程中需要从磁盘读取数据，但通过分布系统将数据分散到不同的服务器磁盘上，提高整个系统的处理能力。目前很多大型互联网和电子商务公司采用基于 X86服务器的分布式集群系统，依靠海量的 X86服务器部署解决高流量并发的问题。

第三代，基于内存的单一系统：将整个数据库放在内存中，计算过程不需要从磁盘读取数据。整个系统的性能取决于单一系统的性能。传统的内存数据库就是这样的系统，对于企业级的应用可以很好地解决访问速度的问题，但面对海量数据或是海量并发访问的扩展性问题就无能为力。

第四代，基于内存的分布式集群系统：GemFire 就是这样的系统，并行计算是其关键技术之一，因而可以通过增加服务器部署规模，在内存计算的基础上，线性扩展性能。

12306之前采用 Unix 小型机架构，采用 GemFire 技术改造成 Linux/X86服务器集群架构，就意味着一下跨越三代。从小型机到大内存 X86服务器集群，不仅让性能提升了一个数量级，而且成本也要低得多。

GemFire 是 Pivotal 企业级大数据 PaaS 平台的一部分。Pivotal 公司的企业级大数据 PaaS 平台主要有三个层次：云基础架构层 Cloud Fabric、大数据基础架构层 Data Fabric、应用开发基础架构层 Application Fabric。GemFire 属于大数据基础架构层，此外，Greenplum 数据库也属于这一层;云基础架构层的技术是 Cloud Foundry;应用开发基础架构层的技术是 Spring Framework 和 RabbitMQ 等。

原文链接：http://www.chinastor.org/GuoNeiXinWen/1588.html?utm_source=tuicool

利用 ElasticSearch 和 Redis 检索和存储信息

本文来自 Zuhaib Siddique 的一次专访，Zuhaib 是群聊 IM 制造商 HipChat 的生产工程师，下面我们一起看 Tod Hoff 的整理。

以下为译文：

如果从企业应用的生存率来看，选择企业团队信息作为主要业务，HipChat 的起点绝非主流；但是如果从赚钱的角度上看，企业市场的高收益确实值得任何公司追逐，这也正是像 JIRA 和 Confluence 这样的智能工具制造商 Atlassian 于2012年收购 HipChat 的原因。

同时，或许你不知道的是，在 Atlassian 资源和人脉的帮助下，HipChat 已经进入了一个指数增长周期。12亿的信息存储意味着他们现在每隔几个月的信息发送、存储和索引量都会翻一番。

如此快速的增长给曾经充足的基础设施带来了很大的压力，HipChat 给我们展示了一个通用的扩展思路。从简单开始，经历流量高峰，然后思考现在怎么办？使用更大的计算机通常是第一个和最好的答案，他们也是这样做的。这给了他们一些喘息空间去考虑下一步怎么做。在 AWS 上，在某一个拐点之后，你开始走向云特性，也就是横向扩展，这就是 HipChat 所做的事情。

然而 HipChat 的发展也并未是顺风顺水，安全性的担忧推动了 HipChat 的云（SaaS）版本之外内部部署版本的发展。

即使 HipChat 没有谷歌那么大规模，我们仍能从中学到好东西，比如他们如何及时索引和搜索十亿信息，这也是 IRC 之类和 HipChat 之间的关键区别。在负载下索引和存储信息，丢失信息是一个艰巨的挑战。

这是 HipChat 选择的路，我们一起展开……

统计

- 每秒60条消息
- 12亿文档存储
- 4TB 的 EBS RAID
- 在 AWS 上8个 ElasticSearch 服务器
- 26个前端代理服务器，是后端应用服务器的一倍

- 18个人
- 0.5TB 的搜索数据

平台

- 主机：AWS EC2 East 上的75个实例全部使用 Ubuntu 12.04 LTS
- 数据库:目前用于聊天记录的 CouchDB,过渡到 ElasticSearch。MySQL-RDS 用于其它的一切
- 缓存：Redis
- 搜索：ElasticSearch
- 队列/Worker 服务器：Gearman（队列），Curler（Worker）
- 语言：Twisted Python（XMPP Server）和 PHP（Web 前端）
- 系统配置：开源 Chef+Fabric
- 代码部署：Capistrano
- 监控：Sensu 和 monit 将警告抽送至 Pagerduty
- 图：statsd + Graphite

产品

- 流量突发。在周末和假期将是安静的。在高峰负荷期间每秒有几百个请求。实际上占用大部分流量的并不是聊天信息，而是状态信息（away、idle、available），人们连接/断开等。因此每秒60条消息似乎很少，但是它只是一个平均水平。
- 通知中心 HipChat，在这里与团队合作，并得到来自工具和其他系统的所有信息。有助于使每个人都在消息圈内，特别是远程办公。
- 使用 HipChat 而不是 IRC 之类,很大的原因是 HipChat 存储和索引每一次对话，以便你以后搜索它们。强调搜索，这个特性的好处是你可以在任何时候做回溯，了解发生了什么和同意了什么。如果在发送一条信息时，你的设备无法访问，它也会将消息路由到同一个用户的多台设备中，并做临时消息缓存/重试。
- 更多的用户带来更快的增长,他们在各个方面使用产品而带来的更多预定,也可以从他们的 API 集成中看到增长。
- 存储和搜索信息是系统中主要的可扩展性瓶颈。
- HipChat 使用 XMPP 协议，因此任何 XMPP 客户端都可以连接到系统中，这

点非常有利于采用。他们已经建立了自己的本地客户端（Window、Linux、Mac、iOS、Android），并带有类似 PDF 浏览、自定义表情符号、自动用户注册等扩展。

- 在以前，将 Wiki 这样的工具引入到企业文化是几乎不可能的。现在，企业级的工具多已在企业落脚，这是为什么？
- 基于文本通信已被广泛接受。我们有短信、IM 和 Skype 的形式，所以现在使用聊天工具是自然的事情。
- 异地工作模式的崛起。团队越来越分散，我们不能只是坐在一起进行一个讲座，一切文档化的需要意味着组织通信将有一笔巨大的财富。
- 增强的功能。把像内嵌图片、GIF 动画等功能做得生动有趣，会吸引更广泛的群体。
- HipChat 有一个 API，这使得它可以编写类似 IRC bots 这样的工具。例如使用 Bitbucket 提交——在10:08开发者 X 提交一些代码来修复一个漏洞。代码发送通过 HipChat 直接连接到代码提交和提交日志，完全的自动化。Bitbucket 提交会击中一个 web hook，并使用一个 addons 来张贴信息。Addons 帮助编写 bots，转入你的 Bitbucket 账户。比如我有我的 API 令牌，我想在每次提交发生时张贴到这个 API 上，工作原理类似 GitHub。
- 在客户端 Adobe Air 启动时，内存泄露会导致宕机，因此将其移动到本地应用上。这是个麻烦，也是机遇。同一个公司中都存在许多跨平台跨部门的用户，你需要站在用户的角度思考。希望用户在所有的系统中都有很好的体验，用户不仅仅是技术人员。

XMPP 服务器架构

- HipChat 是基于 XMPP 协议的，XMPP 节里的内容就是消息，可能是一行文本或者日志输出的长段等等。他们不想谈论自己的 XMPP 架构，所以没有很多的细节。
- 他们没有使用第三方的 XMPP 服务器，而是利用 Twisted Python 和 XMPP 库建立了自己的服务器。这使得可以创建一个可扩展的后端、用户管理，并轻松的添加功能而不用在其它代码库上修改。
- AWS 上的 RDS 用于用户身份验证和其它使用事务及 SQL 的地方。这是一个稳定、成熟的技术。对于内部部署的产品，则使用 MariaDB。

- Redis 用于缓存。信息，如哪些用户在哪些房间，状态信息，谁在线等都是信息。所以，你连接的是哪个 XMPP 服务器并不重要，XMPP 服务器本身并不是一个限制。
- 痛点是 Redis（还）没有集群，因此使用了高可用性的 hot/cold 模式，所以，一个从属节点已经准备就绪。故障转移从主节点到从属节点大概需要7分钟，从属节点的发布是手动的，不是自动的。
- 提高负载可以发现代理服务器中的弱点所在，也可以清楚能支撑多少个客户端。
- 这是一个真正的问题，正如不丢失信息是一个很大的优势。显而易见，不丢失信息比低延迟更重要——用户更愿意晚点接收信息，而不是根本没有信息。
- 使用6个 XMPP 服务器系统运作良好，然而随着连接点的增加，他们开始看到不可接受的延迟。连接不仅来自客户端，还来自 bots 支持他们的程序设计界面。
- 在第一遍的时候，他们分离出前端服务器和应用服务器。代理服务器处理连接，后端应用程序处理的 stanza。前端服务器数量由有效收听客户数量驱动，而不是由信息发送数量驱动。保持那么多的连接打开，同时提供及时的服务是一个挑战。
- 修复数据存储问题之后的计划是调查如何优化连接管理。Twisted 的效果很好，但是他们有很多的连接，所以必须弄清楚如何更好地处理这些连接。

存储架构

- 向 HipChat 发送的消息已达10亿条，同时还在不停增长，他们将 CouchDB 和 Lucene 对存储和搜索信息的解决方案推向极限。
- 认为 Redis 将会是故障点，而 Couch/Lucene 会足够好。没有做合适的容量计划和查看信息增长率。增长速度比他们想象的更快，不应该集中那么多精力在 Redis 上，而应该专注于数据存储。
- 当时他们相信通过增加容量来扩展，向上移动到越来越大的亚马逊实例。他们发现一点，随着不断地增长，他们利用这种方法只能再工作两个月。所以，他们不得不采用其他的办法。
- Couch/Lucene 超过一年没有更新，它不能做分类。这是采用其他办法的

另一个原因。

- 在亚马逊上大约10亿消息的一半是一个临界点。用一个专用的服务器和200G 的 RAM,他们之前的架构可能仍能工作,但在有限资源的云上就不能工作了。
- 他们想留在亚马逊。
- 喜欢 AWS 的灵活性，性能的添加只需要通过租用实例完成。
- 亚马逊的片状。不要把你所有的鸡蛋都放到一个篮子里，如果一个节点出现故障，你必须要处理它，否则一些用户将会失去流量。
- 使用动态模型。可以快速关闭一个实例，并带来新的实例。云原生类型的东西。可以随时关闭节点。关闭一个 Redis 主节点，可以在5分钟内恢复。目前美国东岸分割4个可用地区，但是还没有多区域。
- EBS 只让你拥有1TB 的数据。在遇到之前，他们并不知道这个限制。使用 Couch 时他们遇到了 EBS 磁盘大小限制。HipChat 的数据是0.5TB。为了压缩,Couch 必须将数据复制到有双倍容量的压缩文件中。2TB 的 RAID 在周末压缩过程中遇到了限制，不想使用 RAID 解决方案。
- 不选择亚马逊的 DynamoDB，因为他们创建了一个 HipChat 服务器，在防火墙后面的托管服务。
- HipChat 服务器驱动技术堆栈的决定。私人版是建立在自己主机上的解决方案。某些客户不能使用云/SaaS 解决方案，比如银行和金融机构，国家安全局已经吓坏了国际客户，因此聘请了两名工程师创建产品的安装版本。
- Redis 集群可以自托管，也可以像 ElasticSearch 那样工作在 AWS 上。在内部部署版本中他们使用 MariaDB，而不是 RDS。
- 不能考虑一个完整的 SaaS 解决方案，因为那会是一个锁定。
- 现在过渡到 ElasticSearch
- 移动到 ElasticSearch 作为他们的存储和搜索后端,因为它可以储存他们的所有数据，它是高度可用的，它可以通过简单增加更多的节进行扩展，它是多用户的，它可以通过分区和复制透明的处理节点损失，并且它建立在 Lucene 之上。
- 并不真的需要一个 MapReduce 功能。看着 BigCouch 和 Riak 的搜索（表现一般），但 ES 在 GET 上的表现是相当不错的。喜欢坏了就扔，省去了故障检测。

ES HA 已令他们在系统的坚固性上感到非常有信心。

- Lucene 的兼容是一个巨大的胜利，因为所有的查询都已经兼容 Lucene，因此它是一个自然的迁移路径。
- 客户数据是相当多样的，从聊天记录到图像响应类型的差别也随处可见，他们需要能够快速地直接从12亿文档中查询数据。
- 此举正变得越来越普遍，HipChat 也使用 ElasticSearch 作为他们的 key-value 存储，减少需要数据库系统的数量，从而降低整体的复杂性。既然性能和响应时间都不错，那完全没有不用的理由。 10ms 到100ms 的响应时间。在没有任何缓存的情况下，某些领域仍然超过 Couch。那为什么还要用多个工具？
- 使用 ES，一个节点故障不会引起任何人的注意。在它再平衡时你会得到 CPU 使用率过高的警报，但是系统仍然运行。
- 用8个 ES 去处理流量的增长。
- 基于 Java 的产品 JVM 调整可能非常棘手。
- 要使用 ES，必须有堆空间容量计划。
- 测试缓存。ES 可以缓存过滤结果，这是非常快速的，但是你需要很大的堆空间。虽然8个主机拥有22G 的内存，但还会随着缓存的打开被耗尽。所以如果不需要就关闭缓存。
- 缓存有问题，因为它会遇到内存不足的错误然后失败。集群会在几分钟内恢复，只有少数用户会注意到这个问题。
- 因为网络的不可靠，Amazon 的故障转移也可能存在问题。在集群中可能会引起错误的选举发生。
- 使用 ElasticSearch 会遇到这些问题。原本有6个 ES 节点作为主节点选举运行，一个节点可能会耗尽内存或者遇到一个 GC 暂停并在网络中丢失。那么其他人就不会看到这个主节点，进行选举，并宣布自己是主节点。他们选举架构中的缺陷是他们不需要法定人数。因此就会出现 Split Brain 问题，从而引起很多问题。
- 解决方案是在专用的节点上运行 ElasticSearch 主节点，那么需要做的事情就是成为主节点，从而避免了后续问题。主节点处理分片的分配是完成，谁是主要的，并且完成复制分片分布图。实现再平衡要容易的多，因为主节点可以性

能优良的处理所有的再平衡。可以查询任何节点，并会做内部路由。

- 使用月索引，每个月是一个单独的索引。每个初级索引有8个分片，然后有两个副本。如果一个节点丢失，系统仍能工作。
- 不要把 RDS 移动到 ES 中。需要使用 SQL 的数据一般储存在 RDS/MariaDB 中，典型的是用户管理数据。
- 在 Redis 集群被释放之前，Redis 中大量的缓存是主/从设置。有一个 Redis 统计服务器，处于离线状态。Redis 历史缓存的最后75条消息，用于防止在第一次加载对话时不间断的访问数据库。也有内部状态或快速数据的状态，比如登入用户数量。

常规

- Gearman 用于异步工作，比如 iOS 的推送和传递电子邮件。
- AWS West 用于灾难恢复，一切都会备份到 AWS West。
- Chef 用于所有配置。ElasticSearch 有一个很好的 Chef 手册，轻松上手。像 Chef，因为你可以开始写 Ruby 代码而不是使用 Puppet 风格的 DSL，它也有一个很好的活跃的社群。
- 收购经验。他们现在已经进入公司的核心资产和人才，但 Atlassian 不干扰工作，之所以相信，是有原因的。可以在内部要求，例如，如何扩大 ElasticSearch ，当别人在 Atlassian 需要帮助时，他们可以加入帮忙的队伍。良好的整体体验。
- 扁平的团队结构。仍然是一个小团队，目前大约有18人。两个人在 DEVOPS，少数平台，IOS、Android 的开发人员在服务器端，一个 Web 开发工程师（在法国）。
- Capistrano 用于部署所有的主机。
- Sensu 用于监控应用程序。让你无需监视堆空间 ElasticSearch 节点，然后在没有任何通知的情况下解决 OOM 问题。目前堆的使用率为75%，这正是他们想要的状态。
- Bamboo 用于持续集成。
- 客户端版本还不正规，开发者驱动，有一个临时区域进行测试。
- 集团标志。可以控制哪些群体得到了一个功能、测试特性能及缓慢释放特

性，除此之外还能帮助控制主机的负载。

- 功能标志。有利于 ElasticSearch 部署过程中的保护。例如，如果他们发现一个漏洞，他们可以关闭一个功能，并回去找 Couch。用户不会注意到差别。在 Couch 和 ElasticSearch 之间的过渡阶段，他们都有应用复制到两个存储。
- 新的 API 版本将使用 Oauth，因此，开发人员可以使用 HipChat API 在自己的服务器上部署。有客户使用自己的服务器是一个更具扩展性的模式。

未来

- 未来几个月将会达到20亿条消息，估计 ElasticSearch 可以处理大约20亿条消息。不确定如何处理负载的预期增长。预计要到 Amazon West 以获得数据心更多的的可用性和可能在不同的数据中心投入更多的用户。
- AWS 自动扩展能力
- 移动到语音，私人一对一视频、音频聊天、基本的会议
- 将来可能使用 RabbitMQ 来传递消息
- 与 Confluence 更大的集成。使用 HipChat 聊天，然后使用 Confluence 页面来捕捉细节。

经验教训

1. **企业应用程序是摇钱树。**卖入一个企业是很痛苦的，销售周期长意味着太多的不确定性。但是如果你成功卖出，那就会获得丰厚的利润，所以你应该考虑企业市场。时代在变，企业却可能是滞后的，但是他们仍然采用新工具和新的做事方式，这其中就有机会。

2. **隐私在产品给企业推销时变得越来越重要，**它会直接影响到产品的选择与否。HipChat 正在做他们产品的备用版本，以使那些不相信公共网络的客户满意。对于一个程序员来说，云作为一个平台非常有意义。对于一个企业来说，云可以是魔鬼。这意味着你必须做出灵活的技术堆栈选择。如果你在服务上100%依靠 AWS，那你的系统移动到另一个数据中心将变得几乎不可能。这对 Netfix 也许并不重要，但是如果你想卖入企业市场，它就很重要了。

3. **纵向扩展以获得喘息的空间。**当你等待弄清楚架构中下一步要做什么的时候，可以花很少的钱去纵向扩展，给自己几个月的喘息之机。

4. **选择不会失败的。**HipChat 做出了不会丢失用户聊天记录优先级，所以

他们的架构将这个优先级反映给保存聊天记录到磁盘，在宕掉后系统恢复时会重新加载。

5. **进入本地。**你的客户在许多不同的平台上，一个本地的应用将会提供最好的体验。对于一个初创公司，那是很多的资源，太多了。所以，卖给拥有更多资源的公司在某种程度上是说得通的，这样你可以建立更好的产品。

6. **功能和群组标志做出更好地发布惯例。**如果你可以选择哪些组看到一个功能，如果你能在生产和测试中关闭功能，那么你就不用担心发布新的构建项目了。

7. **选择你真正自信的技术。**ElasticSearch 应对增长的横向扩展能力让 HipChat 很放心，同样也会有一个很好的用户体验，这才是最重要的。

8. **成为该流程的一部分**，你变得更有价值，难以消除。HipChat 作为人和工具之间的天然契合点，也是来编写实现各种有用工作流 bots 的天然点。这使得 HipChat 在企业中有发挥的平台，它使本来不可建造的功能得以实现。如果你能做到同样的事情，那么大家都会很需要你。

9. **AWS 需要在总线上存在一个单独的节点，**这个要求看起来有点荒谬，但是在云环境下却非常重要，因为机器可用信息在第三方目的源中并不可见。如果着眼机架就会发现它经常有一个单独存在的总线插槽，如果其他插槽可用，他就会知道。这样，你就不必去猜测。在云中，软件采用基于原始 TCP 的连接技术和心跳，去猜测另一个节点是否发生故障，从而导致 Split Brain 问题及启用备库时产生数据丢失。这需要时间去演变，到达完全可靠还需要迈一大步。

10. **产品决策驱动堆栈的决定，**HipChat 服务器驱动技术堆栈的决定：Redis 集群可以自托管；不选择亚马逊的 DynamoDB，是因为 HipChat 在防火墙的后面创建一个托管服务。

11. **你需要打开视野。**你需要容量规划，即使是在云中。除非你的架构从一开始就完全是原生云，否则任何架构都会有负荷的拐点，在拐点他们的架构将不再能够处理负载。看看增长速度，项目出来了。会打破什么？你将会做什么？而且不要再犯同样的错误。HipChat 将如何处理40亿条消息？当下还无法知晓。

原文链接：

http://www.csdn.net/article/2014-01-16/2818165-how-hipchat-stores-and-indexes-billions-of-messages?utm_source=tuicool

潜入蓝翔技校二十天，探究蓝翔黑客真正的奥秘

蓝翔技校，这个红遍大江南北的技校以其神秘性而不为人知，与其类似的还有新东方厨师学校。i 黑马分享一篇从知乎日报上转载的一篇亲历蓝翔，潜伏在蓝翔学习的经历，为你们揭开一个真实的蓝翔技校。

2013年4月，我报名进入蓝翔技校，在计算机系网络技术专业学习了近二十天。体验这篇发表在时尚先生杂志上的文章。另外，美容美发系的学习体验，我准备约另外一个哥们写，得等一段时间。

黑进蓝翔

我到达济南的那个中午，天色阴沉，乌云盖住整座城市。

来接站的司机带我上了一辆溅着泥浆的黑色伊兰特轿车。开出济南西站不久，我们上了一条布满碎石与小坑的路。窗外灰尘弥漫，偶尔有渣土车轰隆驶过。司机走错了几次路，我彻底失去了方向感，只记得路上尽是工地、汽配店、小饭馆、批发市场。不知过了多久，眼前出现一片灰色建筑群。车猛一拐弯驶进一座大院，我们到了。

一个小伙子走过来，笑着跟我打招呼，帮我拿了行李。我匆匆打量周围几眼，四月的济南依然景色寥寥，巨大的广场后面蹲着一座方形的大楼，楼前是长长的阶梯，广场两侧栽种的小树没有几片叶子。这里的一切像极了某个县的县政府。

接待大厅里一片冷清，几个中年妇女在吃馒头。我觉得饿，要了一个馒头，

就着白开水吃。大厅的一端是监控室，整面墙上安满了屏幕，看样子，摄像头布满了每个角落。

接我的小伙子叫赵佳，我一吃完，他就说要带我到处转转。我点了根烟，跟着他在校园逛荡。从外面看，这里和内地县城的中学并无二致：外墙贴瓷砖的教学楼，宿舍阳台上挂满衣服。偶尔能看到几个少年聚在一起抽烟，他们的工作服上布满了油污。赵佳跟我闲聊，一个肯定要被人反复问起的问题来了—“怎么想来蓝翔了呢?”

今天，蓝翔技校已尽人皆知。早些年，它的出名是因为电视和广播上频繁直白的滚动广告，但让其声名远扬的是《纽约时报》的一则报道。2009年底，Google等几十家美国公司受到黑客的攻击。两个月后，《纽约时报》刊登了一则报道：

有两所中国教育机构被追查到与一系列针对 Google 公司和其他几十家美国公司的在线攻击有关，其中一所还跟中国军方有密切关系……这两所中国学校是上海交通大学和蓝翔技校……蓝翔，位于中国东部的山东省，是一所由军方支持建立的大型职业培训学校，为军方培养计算机科学人才。

这个消息令我吃惊。在我印象里，蓝翔技校是一个主要针对农村青年学习就业的地方，它培养的是厨师、汽修工人、挖掘机司机、美容美发师，不是黑客。这则消息就像民间科学家造出了载人航天器一样令人难以置信，更难以置信的是，它来自权威的、最具公信力的《纽约时报》。

我特地查阅了有关那次攻击其他的报道，几乎都来自美国媒体。综合起来，它们共同传递的是：有一批顶级黑客出现了，并且他们来自中国。

它们认为这些黑客极度聪明，使用了十几种恶意代码和多层次加密，潜进受攻击的网络内部。更厉害的是，他们还巧妙地掩盖了自己的活动。就连美国网络安全公司 McAfee 的副总裁 Dmitri Alperovitchde 都说：“从未见过如此高水平的加密。在国防工业以外，从来没有商业公司遭到如此复杂的攻击。”

那些报道认定黑客和中国有关的一个理由是，攻击的目标往往极为明确—有利可图或者机密的知识产权。另外一个信息是，黑客试图通过六个台湾的网络地址来掩饰自己的身份，这是中国大陆黑客的惯常策略。

原文链接：http://news.pedaily.cn/201401/20140114359425.shtml?utm_source=tuicool

[评论]全栈工程师到底有什么用

最近国内外都在流行一个词叫 Full Stack，中文翻译过来叫全栈工程师，也叫全端工程师。微博上很多专业人士都在讨论全端工程师，有赞有毁的。我对全端工程师的定义是：掌握多种技能，并能利用多种技能独立完成产品的人。打外比方，全栈工程师就是一个能独立盖一幢10层小洋楼的人，而普通工程师，则是可以和一群人盖一幢摩天大楼的人。

至于要掌握哪些技能，我觉得这个要跟从事的行业与技术方向有关，做互联网的和做软件的是不一样的，即使是做互联网的，后端也可以分为很多种技术流派。

8/2定律在哪都适用，全栈工程师就是掌握20%常用技能的人，但这20%的技能会有80%的几率被用到，剩下那80%不常用的，让我们 Google 吧。

有人说，全栈工程师在中国已经很多年了，他们叫站长。这个说话有点靠谱但又不那么靠谱，我自己也做过站长，深知作为一名站长需要掌握很多种技术。不靠谱的是，很多站长其实并没有真正写过多少代码，而是熟练利用一些建站软件来建站。

全栈工程师的价值

有人说了，你再牛逼，你懂五种技术，你能干五个人的活吗？全栈工程师并不是说一个人能干几个人的活，而是要从多个方面来看这个问题。

全局性思维

现代项目的开发，很少说只用到一两种技术的，特别是移动互联网大潮下。随便一个互联网项目中用到的技术，就会需要用到后端开发、前端开发、界面设计、产品设计、数据库、各种移动客户端、三屏兼容、restFul API 设计和 OAuth 等等，一些比较前卫的项目，可能会用到 Single Page Application、Web Socket、HTML5/CSS3这些技术，还有像第三方开发像微信公众号微博应用等等。

Web 前端也远远不是从前的切个图用个 jQuery 上个 AJAX 兼容各种浏览器那么简单了。现代的 Web 前端，你需要用到模块化开发、多屏兼容、MVC，各种复杂的交互与优化，甚至你需要用到 Node.js 来协助前端的开发。

所以说一个现代化的项目，是一个非常复杂的构成，我们需要一个人来掌控

全局，他不需要是各种技术的资深专家，但他需要熟悉到各种技术。对于一个团队特别是互联网企业来说，有一个全局性思维的人非常非常重要。

像如果是我经手的项目，我肯定会注意到网页优化，也会考虑到 API 来兼容各种客户端，更会考虑到三屏兼容的问题。不会说项目中完全使用 AJAX 而不顾 SEO，也不会为了功能性而忽略访问速度，我会很好的把握这个平衡，因为我知道它们的权重与实现成本。

沟通成本

项目越大，沟通成本越高，做过项目管理的人都知道，项目中的人力是1+1<2的，人越多效率越低。因为沟通是需要成本的，不同技术的人各说各话，前端和后端是一定会掐架的。每个人都会为自己的利益而战，毫不为已的人是不存在的。< p="">

而全栈工程师的成本几乎为零，因为各种技术都懂，胸有成竹，一不小心自己就全做了。即使是在团队协作中，与不同技术人员的沟通也会容易得多，你让一个后端和一个前端去沟通，那完全是鸡同鸭讲，更不用说设计师与后端了。但如果有一个人懂产品懂设计懂前端懂后端，那沟通的结果显然不一样，因为他们讲的，彼此都能听得懂。

创业公司

对于创业公司来说，全端工程师的价值是非常大的，创业公司不可能像大公司一样，各方面的人才都有。所以我们需要一个多面手，各种活都能一肩挑，独挡多面的万金油。对于创业公司，不可能说 DBA 前端后端客户端各种人才全都备齐了，很多工作请人又不饱和，不请人又没法做，外包又不放心质量，所以全端工程师是省钱的一妙招。虽然说全端工程师工资会比一般的工程师会高很多，但综合下来，成本会低很多。

全栈工程师的困境

我讲技术有两个发展方向，一种是纵向一种是横向的，横向的是瑞士军刀，纵向的是削铁如泥的干将莫邪。这两个方向都没有对与错，发展到一定程序都会相互溶合，就好比中国佛家禅修的南顿北渐，其实到了最后，渐悟与顿悟是一样的，顿由渐中来。

如果一个公司不太懂全栈工程师的价值，那么全栈工程师的地位将会很尴尬，

说得不好听一点，全栈工程师就是什么都会，都么都不会。曾经有一次面试，对方问我很基础的问题，我答不上来，我能做出产品，也知道什么是怎么一回事，我也不会犯那些错误，但我就是答不上概念，要考倒我非常容易。所以在应聘面试的时候，有些时候会吃亏，你可能会不如哪些在某一方面钻得很深的人工资拿得高。

由于经常在各种技术穿梭，我会经常忘记代码的语法和一些 API，所以我经常需要去查 API 甚至查语法，我觉得没有 Google 我几乎没法工作。这在某些人的眼里，是技术不够的表现。我记的只是一个 Key，一个如何找寻答案的索引，而不是全部，人脑不是电脑，我不可能要求我能记下所有的东西。

有一次面试官问我一个问题，我说我不知道，但我猜大概是如此这般，对方问我，你为什么这么猜，我说凭直觉，对方笑了笑没说话。面试完后我一查，果然和我猜的差不多。没错，我就是凭直觉，但这种直觉和女人的直觉不一样，这种直觉是技术上的直觉，是你过去技术经验累积的一个反射。

我不是一个非常专业的 Web 前端，也不是一个非常专业的 Node.js 开发工程师，更不是一个非常专业的 iOS 开发工程师。用人单位会问我，你到底是专业做哪一个方面的，我为什么要给你这么高的工资？

有什么资格来谈全栈工程师

我应该算是一个全栈型工程师了，行业经验已经超过10年。独立做过不少产品，也带过不少项目，经过的产品包括桌面端、Web 产品、移动端产品，Web 端涵盖前端与后端，移动端主要做 iOS 和混合开发。

熟悉 Web 前端，对 MVC/模块化开发有实战经验，熟悉 CoffeeScript、Grunt、RequireJS、Handlebars 等等，自己写过小型的 Javascript 框架，一个项目中的 JS 代码超过一万行。熟知网页优化，知道如何让网页变得更加快速。也略懂 SEO，知道什么样的 URL 和代码会更讨好 Spider。

熟悉 Node.js，有几个项目都是基于 Node.js 的，目前发布有开源的 Blog 程序 Purelog，在 NPM 上有多个模块发布。熟悉混合开发，过去我曾经有超过一年的时间是在研究 Hybrid 技术，多个 App 基于混合开发技术，也有开发类似于 PhoneGap 的解决方案。对 HTML5在手机上的表现颇为熟悉，挖过很多的技术坑，如白屏问题，Sqlite 问题，滚动条问题，硬件动画加速、点击延时问题等等。

会做设计，熟悉 Photoshop，所有的产品不管是 Logo 还是界面全都是自己做的设计，虽然在资深的设计师眼里不值一提，但在工程师队伍中算是比较另类了。

熟悉 Objective-C，有两年以上的 iOS 开发经验，在 App Store 上有约十款 App。熟悉服务器的一般性操作，自己有 VPS 并运行多个网站，虽然配置服务器经常要去 Google。

早年曾经做过一年的 Delphi，也曾做过几年的 ASP.net，虽然这些技术我目前已经放弃，完全转向*nix 平台，但累积下的经验是在的。多年的项目管理经验，曾在三个公司担任过项目经理，累计项目管理经验超过4年。在多个科技门户发表过技术和评论文章。

为什么我会成为全栈工程师

我相信很多全栈型工程师会和我一样，是因为要创业才成为全栈型工程师的。我有一颗创业和做产品的心，而且我又是一个不愿意麻烦别人的人，有些人擅长整合资源，空手套白狼，但我显然不是这种人，所以我只好自己做了。

在经历过两次孤独的创业之后，我发现我并不是一个适合独立创业的人，所以，我成为了一个全端工程师。我最初是做 ASP，后来自己创业写客户端用 Delphi，然后写了三年的 ASP.Net，2010年的时候因为公司需要开始做 iOS 开发。Web 前端是一直自己在做，项目中的的 Javascript 基本都是我自己在做。

我热爱写代码，热爱重复发明轮子，热爱新技术，我想这也是我成为一个全栈工程师的重要原因。

于我自己来说，我觉得全端工程师的乐趣要比一般工程师来得多，因为你知道一个产品的形态，你可以去设计一个产品，你是从全局的视野来做事情，你得到的成就感会更多一些。

原文链接：http://www.cnbeta.com/articles/268822.htm?utm_source=tuicool

软件开发中团队首领的好坏之分

软件开发的成败更多的是在于人，而不是技术。当从大学毕业时，我以为，只有精通了各种技术才能成为一名伟大的程序员，以为人件管理技术是经理们的专属领地。但事实却给我好好的上了一堂课。经常我能听到有人说人件管理技术是学不到的，是一种天份，对这种观点我不敢苟同。

没有人能生来就带有某种技能，我们都是通过观察和模仿（我们的偶像）来学习。你也许通过读书得到了这方面的一些知识，但是，我记得一句老话，我十几岁时读过的一本书的封面上印着它："生活从书本中学不来，只能靠自己去感受。"如果你感觉在处理人际交往方面有困难，那么，唯一的方法是去观察，去学习。就这么简单。即使你现在不是一个团队首领，没有带领任何人，学习如何做一个领袖也是值得去做的事，在我们这样一个以人为本的产业里尤其是这样。

虽然跟优秀的团队首领交往能给人非常好的感觉，但糟糕的团队首领却能凸显出领导能力的真正价值。我很幸运有过这样的经历，我有机会接触了不少十分糟糕的团队首领——你可以想象他们有多糟。下面我们就来看看好的团队首领和不好的团队首领之间的对比。

内容	好团队首领	差团队首领
责任感	好的团队首领勇于承担责任。如果项目失败，他会认为是自己受首先应该受责备的人，他有勇气承认这些。	差的团队领导会认为不是自己的错，他会把所有精力都投入到证明团队其他人员有罪，或把责任推到团队中某些他不喜欢的人身上。

努力工作	团队首领应该是团队成员的劳动模范。至少会像团队其他成员一样努力工作。并不是因为他是带队的，他就有权了只选择干自己喜欢干的事情，把不好干的活儿丢给其他人。	差的团队首领认为自己有更重要的事情。既然手下有这么多人可以用，干嘛还要自己去编码呢？
指导	好的团队首领会指导培养团队中的初级程序员。他不会让这些人在有难度的任务上受挫折。他知道对团队成员的培养投入必定能在开发质量上获得回报。	差的团队首领不在乎这些。经验少的程序员就应该用高难度的任务来锻炼。
尊重	好的团队首领尊重所有团队成员，无论他们的技能如何。他知道带领一个团队的唯一方法是靠获得尊敬，而不是恐吓。	差的团队首领除了自己外不尊敬其他任何人。当有人出错时他会嘲笑，而且会把这些事情写入对上级领导的报告中。
晋升	好的团队首领相信技术和专业能力的价值。他相信能通过做好自己的工作来获得应该得到的职位。	差的团队首领技术不行，拍马屁很在行。对下属趾高气扬，对上司极力奉承。

情绪控制	好的团队首领性格成熟，他知道如何控制自己的情绪。他不会对着团队成员大喊大叫，也不会说出有任何威胁性的话。	差的团队首领喜欢亮出自己的身份，认为恐吓是管理一个团队最好的方法。他认为恐惧是最好的动力。
信任	好的团队首领信任他的团队成员。他知道这些同事都是用知识技能干活的人，这些知识技能需要提高的。这就是为什么他会鼓励所有人走出自己熟知的知识领域，用一些新技术，这样他们会学的更多，变的更强。	差的团队首领除了自己不相信任何人。那些经验不足的人就只能去写文档，或去给自己的代码写单元测试。毕竟测试这种活儿没人愿意干。
任务分配	好的团队首领会选择那些谁都不愿意干的活儿。把最艰苦的任务分给自己，身先士卒。	差的团队首领总是挑选自己喜欢干的活。也许是一个他一直想尝试的新框架。为什么这么好的机会没人跟我争？当他发现框架太复杂后，他会把它丢给团队其他成员，自己去修改一些小问题。
报告问题	好的团队首领会努力解决所有问题。但总有不能完全做好的事情，这个时候，他会立即将情况报告给上级，让上级采取正确的措施。	差的团队首领总是掩饰问题。他不喜欢报告出现的问题，因为这会影响到他的声誉。如果问题出现，他会找一个人出来顶罪，永远不是他的责任。

代码审查	好的团队首领喜欢代码审查，鼓励团队成员参与代码审查。当有重复出现的问题时，他会把问题记录到知识库里，这样所有人就能通过知识库更好的解决遇到的问题。	差的团队首领没有时间做代码审查，每个人都各自做自己的事。如果有人的代码出了问题，差的团队首领只告诉他如何解决问题。
挫折	一个优秀的团队首领也许有一个糟糕的上级领导，但他告诉自己绝对不能像那个家伙那样为人处世。他很成熟，已经知道如何从别人的错误中吸取教训。	差的团队首领在自己受挫时希望下属也遭受跟自己一样的痛苦。
新思想	好的团队首领善于聆听。他会让团队成员举行各种头脑风暴。他知道好主意往往在无意间冒出来。	差的团队首领不喜欢别人“炫耀”他们自认为的好想法。他认为自己的想法更好。如果他听到了一个有趣的观点，他会取笑它，然后到上级领导面前邀功说自己有了一个好主意。

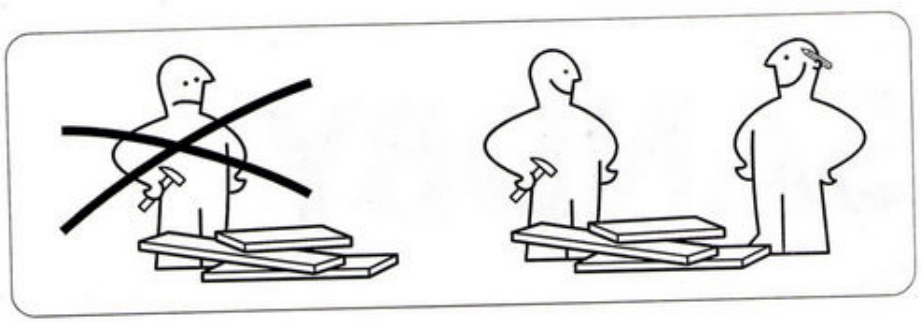
不称职的团队首领会让公司破财。如果项目由一个不称职的人带领开发，最终弥补他的过失的成本会比让一个有水平的团队重新开发还要高。这听起来像是童话，但上面这些差的团队首领的特征都是从真实生活中整理出来的，我很感谢所有这些不称职的人，他们让我学到了团队领导艺术中难得的教训。

原文链接：http://blog.sae.sina.com.cn/archives/2518?utm_source=tuicool

学编程就像选家具：去宜家还是从种树开始？

在日常生活中，常常会听到："我想学编程"，看似简单的一句话，仔细分析，其中的奥妙可不少。你想学什么样的编程，是一个简单的 Hello World，还是开发移动应用呢？本文作者 Scott Hanselman 把这个问题比喻成选家具，是去宜家还是从种树开始呢？并且与大家探讨了码农、黑客、程序员、开发者和计算机科学家之间的区别？下面是笔者对原文的翻译。

最近有朋友向我提问：说他**想学如何编写代码，但不知道该如何下手以及该从哪里起步？**



学习如何编码——是去宜家还是从种树开始？

好比想当木工，你可以选择从种树开始，然后砍树、打磨，最终制作家具。或者，你也可以直接去宜家，也可以介于这两者之间。

直接修改 WordPress 主题就好比是去宜家，自己动手编写一个 Web 框架就好比种树，通常都是因为没有自己喜欢或者现成的"树"。**无论是自己"种树"还是直接选择现成的，你都必须决定自己想要的家具。**

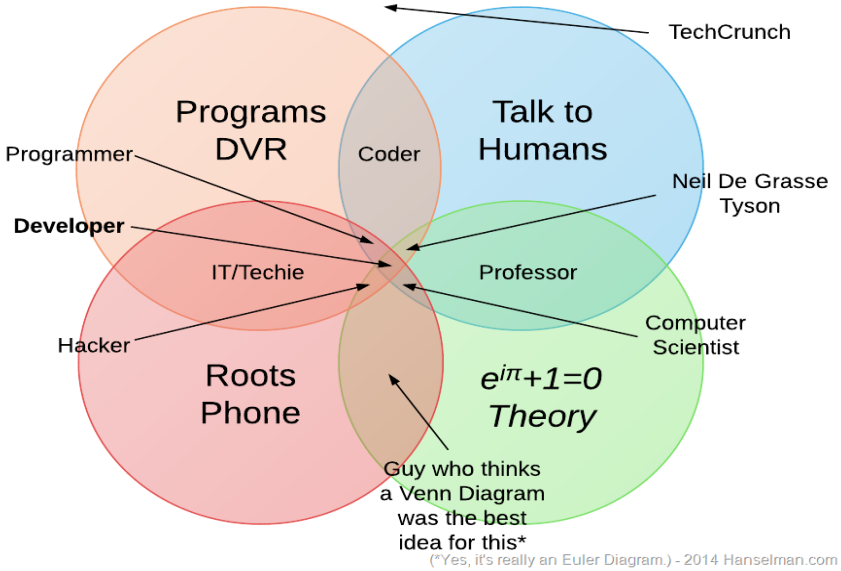
好吧，该从哪里开始？

每当我听到有人想学编程时，我都非常高兴，下面这些网站都是非常不错的学习资源：

- Codecademy.com
- KhanAcademy.org/cs
- Code.org
- learncodethehardway.org
- Udacity

- CodeSchool
- Harvard's CS50x at edX
- CoderByte

码农、黑客、程序员、开发者和计算机科学家的区别？



你可以认为这些词的意思都一样。也许你还听过 Geek、nerd 和 dweeb，但知道他们的不同还是非常重要的。了解这些不同你就可以清楚地知道你到底所属哪一个层次：

- **码农**：通常是发现问题并解决的人，但解决方案并不完美；
- **黑客**：通常是底层专家，技术过硬，曾深入某领域研究，并且非常精通；
- **程序员**：写代码并了解算法，经常独自专注地工作；
- **开发者**：是真正厉害的多面手，熟悉多种系统和语言，可以熟练交叉使用。知识广泛的专业人士，有良好的沟通和团队协作能力；
- **计算机科学家**：知道并了解计算机的工作原理，精通理论层面，数学达人。

如果你已经接近其中的一个层次，你可以想想今后应该朝哪个方向迈进。

假设是 Web 编程？

在几年前，如果有人告诉你想学编程，你可能会编写个 Hello World 程序，或者安装下 Visual Basic，拖一个 Button 控件，然后用消息框弹出个 Hello World。

这或许就是大家在入门时遇到的第一个程序，作为 JavaScript 和 Web Service 入门，我认为这是很好的开端。但关键是，并不是所有的应用程序都是

Web 应用程序。应用程序会使用来自服务端的数据、发送通知、文本、Email 和 Tweets 等信息，甚至是一个非常小的应用程序，它也有可能从 Web 服务器上调用一些数据。每个应用程序都会实现相应的功能，并且应用在相应的地方。所以你要搞清楚，你所说的编程是指 Web 编程还是其它编程。

现在，如果你想成为一名程序员，或者更加精准地说，你想成为一名高效的 Web 程序员，那么你就想弄清楚，当用户在网页里输入 twitter.com 时，浏览器到底做了些什么，原理是什么？如果你想成为一名木匠，你就想了解树是如何长成，如何挑选好的木材；如果你想成为一名赛车手，你就想知道引擎的工作原理；亦或者是，如果你想成为一名管道工，你就必须知道水源。

你弄清“我想学习编码”的真正含义了吗？

问题的根本所在是你到底想从事哪方面的代码编写？网站开发、网站设计、还是编写移动应用程序、还是想编写一个小工具、这些都是完全不同的终点，并且它们都会有很好的入门教程，如果你想深入进去。

- 对 Web 开发感兴趣？

任何相关“Learn to Code”的网站都非常不错

- 对硬件感兴趣？

可以考虑 Raspberry Pi 或 Arduino

- 对你每天所使用的代码/历史代码很感兴趣？

可以去阅读 Charles Petzold 的“Code”

- 已经通晓一定的技术但想走的更远？

订阅 http://learncodethehardway.org

- 想成为一名很好的多面开发者？

阅读 Mike Gunderloy 的“Coder to Developer”

写在最后

对于想学编程的人，他们完全可以从 Web 编程开始，学一点 JavaScript 然后开始编写 Web 应用程序。但如果你的兴趣愈加浓厚，你也可以钻研一些不同的编程领域，给自己增加更多锻炼的机会和发展空间。

原文链接：http://www.csdn.net/article/2014-01-14/2818134-Learn-program?utm_source=tuicool

龙泉寺：如何用互联网思维管理一家寺庙？

新年新气象，很多人都开始踏上了新的人生旅程。据传一位中科院的计算机博士也在新年伊始突破人生——加盟北京龙泉寺，成为一个禁欲的和尚。网上热传的段子说：“一个中科院的博士，好不容易攒够了论文，结果导师出家了……没有人知道导师人现在哪里，但偶尔还是会回一两封邮件。他发邮件说，老师，我要答辩。导师回，施主，凡事都不要太执着，答辩不答辩，其实都是空……”

记者多次前往龙泉寺探访，采访了该寺多位法师和居士。他们澄清，龙泉寺最近剃度的法师中并无中科院博导。龙泉寺的确有几位毕业于北大、清华等名校的高学历法师，但也有初中毕业生，“清华北大分校”的名号乃外界炒作，过分夸张。

不过仔细研究龙泉寺的高僧名录，你会发现龙泉寺的科研实力不容小觑。

你知道龙泉寺的科研实力有多强吗?

一、天下极客出龙泉

龙泉寺位于北京海淀区西北边，凤凰岭自然风景区内，座落在北京西山凤凰岭山脚下，始建于辽代应历初年，距今已有一千多年的历史。几经损毁又几度重建，2005年这里重新恢复开放为佛教道场。古刹背山而建，到城区只有一班公交车。

时光倒回2011年11月，一年一度的中国移动开发者大会，龙泉寺贤信法师的一身僧衣和淡然寂静的表情吸引了其他参会者的视线。从此，中国玩互联网的留下一个传说：龙泉寺，有极客。

据说几年前，微信之父张小龙有一次入京到龙泉寺散心，心中关于微信的产品困惑久久不得解。无聊中，张小龙与寺中扫地僧攀谈起来，发现对方居然懂得技术和产品，深入聊天之后，张小龙震惊于对方的才学与见识，虚心请教，之后闭关七天回到深圳，微信终于大成。

互联网圈百晓生、车库咖啡创始人苏菂谈到自己对龙泉寺僧人的印象说:“他们都是工程师出家，是高才生写代码”。

古语有云：天下武功出少林，牛叉极客入龙泉。你知道龙泉寺的科研实力有多强吗?龙泉寺部分高僧名录摘录如下：贤威法师，龙泉寺管理委员会秘书，中科院生物物理研究所博士;贤启法师，龙泉寺管委会的五位成员之一，清华大学核能和热能物理博士;2010年出现在报道上的第47届国际数学奥赛金牌、北大数学系高材生柳智宇，在龙泉寺清修3年后，已经正式剃度成为法师，法号贤宇，目前在寺内负责校律工作，主要是整理编校一些佛教典籍……

龙泉寺2014年科研项目是这样的:《大数据时代云计算推动沙门信息化研究》、《基于社会网络的西方八百罗汉关系研究》、《基于文本数据挖掘的梵文分词研究》、《大数据时代的佛家信息管理》、《论 SNS 在各法门寺弟子交流之间的应用》……

龙泉寺坐落在北京西山凤凰岭脚下，始建于辽代。而今龙泉寺闻名全国，并不靠高大的庙宇，而是有一些高学历人才到此出家。

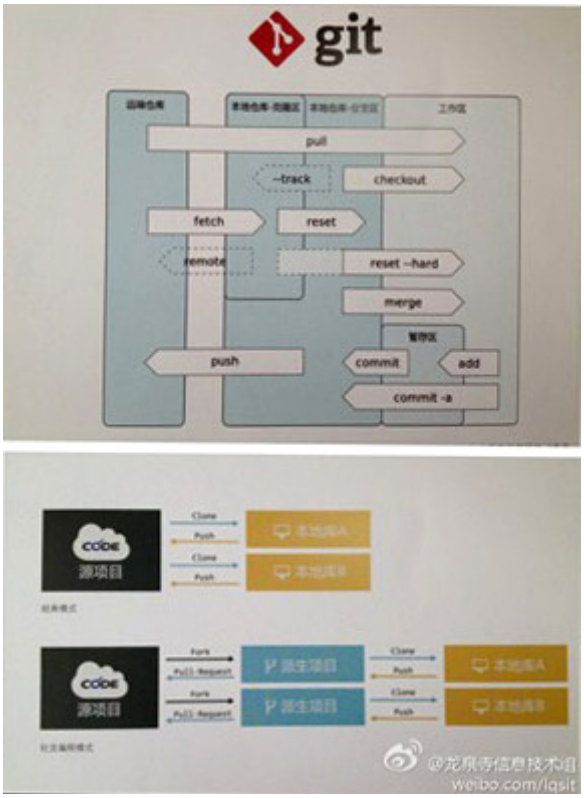
二、自建信息化管理系统

龙泉寺的日常活动是这样的，龙泉寺信息技术组除了开发软件，还有周末例会、集中开发、IT 培训、有机农场劳动、栗子园采摘、登山远足等活动。

龙泉寺的贤信法师，精通 Linux 和 Mongodb，带领一帮 IT 义工，搭建了专业的信息平台，自己编写代码，用数字技术管理寺里的日常生活及与外界的联系等一切事宜。

龙泉寺还有自己的藏经阁。生于1984年的贤才法师毕业于大连理工大学，现在更像一位专业的图书馆工作人员。2012年7月，国家图书馆的工作人员来龙泉寺访问，贤才法师负责接待，双方讨论了“慧海佛教百科”数据库的使用和改进。

贤才法师和他的助手们采取了一套相当靠谱的编目管理流程，对文献资料进行分类，编制目录，建立馆藏目录体系。这一流程被分成15个步骤，包括“记到”（对到馆图书进行信息核对及登记）、“查重”（对馆藏已有书目的筛选工作，通常保留3本）、给分类号、贴登录号、编 MARK 数据（即书目数据，含书名、作者、分类、简介、索书号等）。图书上架之后，寺内的法师可以前来借阅，他们拥有自己的电子借阅卡。



龙泉寺信息技术组新浪微博：“……在曹敬波家复习 Git。SVN、CVS、Git、GitHub……，从版本控制的种类、渊源，到 git 具体命令，云涛边讲、边演示，其他人边听、边操作。两个多小时很快过去，大家意犹未尽，相约尽量多些线下学习……”

三、禅悟与编程

贤信法师比贤才法师年长些，高高瘦瘦，走路和说话的时候很专注。

“我是北工大计算机专业的，毕业后做过几年程序员。后来不是很喜欢这个专业了，因为变化太快，心脏受不了。”

2009年初，他正式出家，法号贤信。在问起出家的缘由，贤信法师不肯多言，只大略地说是“因缘所致”。

出家之后，贤信再度捡起了信息技术。他注意到，客堂每天都要处理挂单和床位的各项事务，EXCEL 只被当成记事本来用，没有发挥出自动计算和归类的功能。他便想，如果有一个数据库，信息准确，管理有条理，那该多好啊。

2010年春节，贤信法师一个人开发完成了龙泉寺的“挂单系统”。

为了一些不太懂的技术，贤信着急过。被学诚法师知道了，说他这是“向外用心”——事情没做，已经预设了很多问题。这句话之于贤信，犹如暮鼓晨钟，他开始考虑如何安坐于“不安”之上。

“以前我总想着有一个彻底的解决办法，在这个前提下去落实和推进，但是实际上不存在这种完美的状态，我要学着慢慢去解决。”

龙泉寺信息技术组在新浪微博的签名是：“穿越技术人生，探索终极价值。”某次，组内活动后，信息小组的 PPT 被贴到了网上，其中一篇演讲的题目令人印象深刻，叫做“前端代码之禅”。

原文链接：http://www.yixieshi.com/it/15680.html?utm_source=tuicool